University of Zurich[UZH]

Department of Informatics

FlexiView: A Physics-Based Focus+Context Navigation Technique for Requirements Modeling Tools

Dissertation submitted to the Faculty of Business, Economics and Informatics of the University of Zurich

to obtain the degree of
Doktor / Doktorin der Wissenschaften, Dr. sc.
(corresponds to Doctor of Science, PhD)

presented by
Parisa Ghazi
from Iran

accepted on the recommendation of
Prof. Dr. Martin Glinz
Prof. Dr. Kurt Schneider
Prof. Dr. Dimitris Karagiannis

University of Zurich[UZH]

The Faculty of Business, Economics and Informatics of the University of Zurich hereby authorizes the printing of this dissertation, without indicating an opinion of the views expressed in the work.

Zurich, October 24, 2018

Chairman of the Doctoral Board: Prof. Dr. Thomas Fritz

Look up at the stars and not down at your feet.
Try to make sense of what you see,
and wonder about what makes the universe exist.
Be curious.

—Stephan Hawking

(Physicist, 1942-2018)

Acknowledgement

Several years ago, I embarked on a PhD and I knew that the path that I just stepped into was going to be hard and stressful. However, the amazing people that I had close by me during this journey transformed this experience into the most pleasant period of my life.

First and foremost, I would like to express my sincere gratitude to my advisor Prof. Dr. Martin Glinz for the continuous support of my PhD study since the very first moment till the very last moment, for his patience, motivation, encouragement, and immense knowledge. His guidance for developing a vision of what may come next and his help during all the planning, research and writing was invaluable. I could not have imagined having a better advisor and mentor for my PhD study.

I would also like to thank Prof. Dr. Kurt Schneider and Prof. Dr. Dimitris Karagiannis for being part of my PhD committee as external examiners and for dedicating their valuable time to evaluate my work.

During a PhD there are many ups and downs. It would not have been possible to make it through all the downs and celebrate the ups without my fellow members of the RERG group. It was an awesome experience to work with you all. A lot of thanks to Norbert Seyff, Dustin Wüest, Irina Koitz, Eya Ben Charrada, Martina Kolpondinos-Huber, Sofija Hotomski, Emitzá Guzmán, and Melanie Stade. I appreciate the fun times we had in the office and on conference trips and the valuable feedback they provided on my work. I would like to especially thank Dustin and Irina for sparing their valuable time and sharing their clever ideas generously, without whom I would have felt lonely.

In addition to the research, it was a lot of work to develop tools and experiment with them. This amount of work was not possible without the help of the students who contributed to the development of FlexiView and ImitGraphs and the user studies: Arlind Bejta, Tenzen Rabgang, Nico Strebel and David Lay.

Furthermore, I want to thank my parents and my sisters for their spiritual support and encouragement during my PhD. Last but not least, I want to thank my husband Alireza who was present at every single step I took. We went through all the tensions and reliefs together and shared the stress of hard working and the happiness of success.

Parisa
Zürich, July 2018

Abstract

Requirements engineers capture their system of interest in multiple artifacts and use software tools for creating, modifying and viewing their artifacts. Therefore, they invest a significant part of their time in working with such tools. As a result, the performance of requirements modeling tools affects the overall performance of the requirements engineering process. A performant tool requires both powerful features and a high level of usability. Our focus in this thesis is on the usability of requirements tools.

Usability of requirements modeling tools has not received a notable consideration both in science and industry. As a result, the literature does not provide an overall picture of the usability challenges in such tools and the user interface of requirements modeling tools has not changed significantly for decades. This is because of the lack of knowledge about the usability challenges in requirements tools and the high cost of conducting usability tests.

Our approach to the problem of usability challenges in requirements tools comprises two parallel, interwoven tracks: (i) identifying the

challenges and designing a navigation technique to solve a subset of these challenges, and (ii) creating a testing platform such that fair and fast comparisons of different solutions become possible. Finally, we use the result of the second track to evaluate the result of the first track.

In the first track, we conducted an exploratory study to create an overview of the existing usability challenges in requirements tools and the relevant characteristics of artifacts and screens. Our study revealed that requirements artifacts are often large and interconnected, nevertheless the tools only provide traditional navigation techniques. Thus, practitioners face different challenges and even prefer to use general-purpose tools and non-software workarounds. We devised FlexiView, a physics-based focus+context navigation technique for navigating in large artifacts and visualizing the interconnection between artifacts.

In the second track, we developed an experimental tool for performing fair comparisons of navigation techniques. Our experimental tool (i) has a similar user interface to existing requirements modeling tools, (ii) allows the creation and integration of alternative user interface techniques, (iii) supports ImitGraphs, our extended node-and-edge graphical notation that can imitate the behavior of other graphical notations, and (iv) is capable of logging user interactions precisely. These properties allow quick integration of new techniques while preserving the generalizability of the results.

Employing our experimental tool in controlled experiments for comparing FlexiView with zooming+scrolling, we found a great

potential in FlexiView for improving the efficiency, effectiveness and user satisfaction of performing requirements tasks that involves creating, modifying and viewing requirements artifacts.

Zusammenfassung

Anforderungsingenieure dokumentieren ein zu spezifizierendes System in verschiedenen Artefakten und verwenden Softwarewerkzeuge zum Erstellen, Ändern und Betrachen dieser Artefakte. Daher investieren sie einen erheblichen Teil ihrer Zeit in die Arbeit mit solchen Werkzeugen. Dadurch beeinflusst die Leistung von Werkzeugen zur Anforderungsmodellierung die Gesamtleistung des Requirements Engineering Prozesses. Ein performantes Werkzeug benötigt sowohl leistungsstarke Funktionen als auch ein hohes Maß an Benutzbarkeit. Der Fokus dieser Dissertation liegt auf der Benutzbarkeit von Anforderungswerkzeugen.

Die Benutzbarkeit von Anforderungswerkzeugen hat weder in der Wissenschaft noch in der Industrie eine deutliche Berücksichtigung erfahren. Deshalb bietet die Literatur kein Gesamtbild über die Herausforderungen im Hinblick auf die Benutzbarkeit solcher Werkzeuge, und die Benutzeroberflächen von Werkzeugen zur Anforderungsmodellierung haben sich seit Jahrzehnten nicht bedeutend

verändert. Gründe dafür sind fehlendes Wissen über die Herausforderungen betreffend der Benutzbarkeit, sowie die hohen Kosten für die Durchführung von entsprechenden Tests. Unser Lösungsansatz für das Problem der Benutzbarkeit von Anforderungswerkzeugen umfasst zwei ineinander verflochtene Arbeitspakete: (i) das Identifizieren der Herausforderungen und der Entwurf einer Navigationstechnik, um eine Teilmenge dieser Herausforderungen zu lösen, und (ii) die Entwicklung einer Testplattform, welche faire und schnelle Vergleiche verschiedener Lösungen ermöglicht. Schließlich verwenden wir das Ergebnis des zweiten Arbeitspakets, um das Ergebnis des ersten Arbeitspakets zu evaluieren.

Im ersten Arbeitspaket haben wir eine explorative Studie durchgeführt, um einen Überblick über die bestehenden Herausforderungen in der Benutzbarkeit von Anforderungswerkzeugen zu gewinnen und die entsprechenden relevanten Eigenschaften von Artefakten und Bildschirmen zu dokumentieren. Unsere Studie hat gezeigt, dass Anforderungsartefakte oft groß und miteinander verbunden sind. Trotzdem bieten die Werkzeuge nur traditionelle Navigationstechniken an. Dies stellt Praktiker vor verschiedene Herausforderungen, und sie bevorzugen sogar den Einsatz von Universalwerkzeugen und nicht-softwarebasierten Behelfslösungen. Als Antwort auf diese Herausforderungen haben wir FlexiView entwickelt, eine auf einer physikalischen Metapher basierende Technik zum Navigieren in großen Artefakten und zum Visualisieren der Verbindungen zwischen Artefakten.

Im zweiten Arbeitspaket haben wir ein experimentelles Werkzeug entwickelt, um faire Vergleiche von Navigationstechniken

durchführen zu können. Unser experimentelles Werkzeug (i) hat eine ähnliche Benutzeroberfläche wie existierende Anforderungsmodellierungswerkzeuge, (ii) ermöglicht die Erstellung und Integration von alternativen Techniken für Benutzeroberflächen, (iii) unterstützt ImitGraphen, unsere erweiterte Knoten-und-Kanten-Grafiknotation, die das Verhalten anderer grafischer Notationen imitieren kann, und (iv) ist in der Lage, Benutzerinteraktionen präzise zu protokollieren. Diese Eigenschaften ermöglichen eine schnelle Integration neuer Techniken unter Beibehaltung der Generalisierbarkeit der Ergebnisse.

Beim Einsatz unseres experimentellen Werkzeugs in kontrollierten Experimenten zum Vergleichen von FlexiView mit Techniken zum Zoomen und Scrollen haben wir festgestellt, dass FlexiView großes Potenzial hat zur Verbesserung der Effizienz, Effektivität und Benutzbarkeit bei der Durchführung von Requirements Engineering Aktivitäten.

Contents

Abstract **vii**

Zusammenfassung **xi**

1 Synopsis **1**
1.1 Introduction 1
1.2 Background and Motivation 7
1.3 Research Goal, Questions and Methodology 17
1.4 An Overview of FlexiView 24
1.5 An Overview of ImitGraphs 28
1.6 Contributions 34
1.7 Road Map and Chapter Summary 38

2 Visual Characteristics of Requirements Artifacts and the Challenges of Working With Them **45**
2.1 Introduction 47
2.2 Definition of Terms 51
2.3 Research Methodology 54
2.4 Key Findings 64

2.5 Consolidation of Findings 98
2.6 Existing Work Addressing the Challenges 107
2.7 Related Work 116
2.8 Threats to Validity 118
2.9 Conclusion and Future Work 122
2.10 Acknowledgements 124

3 A Physics-Based Focus+Context Presentation and Navigation Technique for Requirements Artifacts 125
3.1 Introduction 127
3.2 Research Goals 128
3.3 Related Work 129
3.4 FlexiView: A Magnet-Based Visualization Approach 130
3.5 Conclusions 137

4 An Imitating Graph for Faster Usability Tests of Requirements Modeling Tools 139
4.1 Introduction 141
4.2 Related Work 143
4.3 Graphical Models and Graphs 144
4.4 Our Approach 151
4.5 A Sample Usage Scenario 160
4.6 Conclusions and Future Work 164

5 Common User-Interface Features of Requirements Modeling Tools 169
5.1 Introduction 171
5.2 Approach 172
5.3 Results 173

5.4 Choosing the Requirements 175
5.5 Conclusion and Future Work 176

6 FlexiView Experimental Tool 179
6.1 Introduction 180
6.2 FlexiView Tool 182
6.3 Conclusion and Future Work 185

7 An Experimental Comparison of Two Navigation Techniques for Requirements Modeling Tools 187
7.1 Introduction 189
7.2 UI navigation techniques 191
7.3 Study Goals 196
7.4 Language and Tool 197
7.5 Experiment Tasks 201
7.6 The Experiment 207
7.7 Results and Discussion 211
7.8 Threats to Validity 226
7.9 Conclusion and Future Work 228

8 Conclusion 231
8.1 Revisiting the Research Questions 232
8.2 Revisiting the Thesis Statement 235
8.3 Future Work 236

Bibliography 241

A Publications 263
A.1 Conference Papers 263
A.2 Journal articles 265

A.3 PhD Symposium 265

List of Tables

1.1 Joint Types 32
1.2 Connection Types 32
1.3 Node Types 33

2.1 The percentages of different categories of collaboration tools and examples for each category 74
2.2 Challenges of working with artifacts that are larger than the available screen. Prioritization is explained in the text. 75
2.3 Challenges of working with multiple artifacts at the same time. Prioritization is explained in the text. 82
2.4 Challenges of not storing interrelationship information properly. Prioritization is explained in the text. 84
2.5 Effectiveness of storing interrelationship information 86
2.6 Alternative techniques to maintain overview 91
2.7 Distribution of methods of storing artifact relationship information. 96
2.8 Key Findings 99

4.1 Joint Types 162
4.2 Connection Types 162
4.3 Node Types 163

5.1 Basic actions and how they can be performed in tools174

List of Figures

1.1 The phases of this research, their corresponding research questions, the steps of the engineering cycle, the problem type and the chosen research method of each phase 20
1.2 The working mechanism of FlexiView: (a) A given graph. (b) A virtual magnet is put on the upper-left node. (c) the strength of the magnet is increased. (d) A second magnet is put on the lower-right node. (e) The strength of the second magnet is increased. (f) How the user actually sees the graph. 26
1.3 A comparision between zooming and FlexiView. (a) A given graph in which the node of interest is indicated with a red arrow. (b) The node of interest is zoomed in. (c) The node of interest is magnetized using FlexiView. 27
1.4 Parts of three different graphical models and their equivalent ImitGraph: (a) activity diagram, (b) class diagram, and (c) goal model 29

1.5 An activity diagram, its corresponding ImitGraph, and pair-wise corresponding constituents of them. . 32
1.6 Three steps of two equal tasks in the activity diagram notation and the ImitGraph notation. 34
1.7 The relations between contributions and the research questions they correspond to. 35
1.8 The relations between chapters, contributions and research questions. 39

2.1 Distribution of participants with regard to (a) distribution of the roles (b) years of experience and (c) size of the company measured in number of employees 59
2.2 Geographical distribution of participating companies 61
2.3 Demographic information about the survey participants: (a) distribution of the roles, (b) the level of experience for academic participants measured by their academic degree, (c) years of experience in working with artifacts, (d) size of company for industry practitioners measured in number of employees 63
2.4 Percentage of software engineering graphical artifacts that fit on the participants' screens 66
2.5 Distribution of screen sizes of the participants . . . 68
2.6 Percentages of graphical artifacts that fit on a 28-inch screen 69
2.7 Number of artifacts used at the same time 70
2.8 The distribution of graphic notations used by participants 71
2.9 Overall and role-wise distribution of the average artifact size (S_μ) 77

2.10 Ranking of the participants with respect to artifact and screen size used. The ranks of the participants are determined by measuring their Euclidean distance from the reference point. 78

2.11 Ranking of the participants with respect to artifact size, screen size and number of artifacts used. The ranks of the participants are determined by measuring their Euclidean distance from the reference point in the three-dimensional space. 81

2.12 Ranking of the participants with respect to artifacts used at the same time and the effectiveness of the method they use. The ranks of the participants are determined by measuring their Euclidean distance from the reference point. 87

2.13 The percentages of the methods that participants used to handle artifacts that are larger than the screen 90

2.14 The percentages of the methods that participants used to handle multiple artifacts at the same time 92

2.15 The effort needed to setup and maintain different methods of storing interrelationship information. The corresponding value for “Software tool” is emphasized by a horizontal dotted line. 97

2.16 Participants’ overall satisfaction with the available methods to store artifact interrelationship information on a five-point Likert scale. 98

2.17 The pattern used to create the chains that are building blocks of Figure 2.18 101

2.18 A consolidated view of all the findings of this paper. The diagram consists of several chains starting from a property and ends with a usability factor. The pattern of the chains is depicted in Figure 2.17. A full description of the chains and their relations to the findings is available in Section 2.5. 102

3.1 (a) A sample of regions modeled by metal balls and springs. (b) The positions of the balls are determined by three forces: the Spring Repulsive Force (SRF), the Spring Attractive Force (SAF), and the Magnet Repulsive Force (MRF). 132

3.2 (a) RE artifacts and their regions. (b) The user has placed a magnet in the top right region, resulting in the enlargement of this region and the appearance of more details. In the shrunk region at the bottom left, fewer details are displayed. (c) The user has increased the strength of the magnet, so the corresponding region grows and the other ones shrink. 133

4.1 A sample scenario of drawing an activity diagram based on a natural language description of the underlying process 147

4.2 Inserting a decision node between two activities in two ways: (i) removing the connection, creating the decision node and connecting them together, and (ii) creating the decision node, dragging it over the connection and the tool automatically breaks the connection into two connections. 148
4.3 Inserting a node between two existing nodes in a simple graph can be done differently from the similar example of Figure 4.2 150
4.4 Similar tasks on an activity diagram and its equivalent graph may result in different layouts 151
4.5 A node and a connection of a specialized graph . . 152
4.6 Parts of "a" an entity-relationship diagram and "b" an activity diagram, and their equivalent specialized graphs 155
4.7 Examples of specialized graph commands that instruct participants how to manipulate diagrams: (a) Create a new node, (b) Branching from an existing node, (c) finding a node that is not referenceable, (d) finding a connection, assigning it a label and inserting a new node in the middle of it. 157
4.8 Equivalent scenario of Figure 4.1 in ImitGraph notation. The first column contains the task given to the participants. The second column shows the detected model-space operations by the participant. The third column shows the corresponding tool-space operations. The last column shows the resulting diagram at each step. 165

6.1 Three screenshots of the tool while showing a simple diagram (a) and while the upper-left node is zoomed in (b) and magnetized (c). 184

7.1 FlexiView's magnifying process: (a) A graph. (b) A magnet is put on the lower left node. The green lines (not visible to users) show how FlexiView partitions the space. (c) The power of the magnet is increased, causing the focused node to enlarge and other nodes to shrink. (d) How the magnified graph is actually displayed (without the green lines). 194
7.2 ImitGraph notation: (a) a node. (b) a connection with two joints. (c) a sample activity diagram. (d) an ImitGraph equivalent to the activity diagram. 198
7.3 User interface of the tool used in the experiment. . 201
7.4 The ImitGraph used in task Search, representing a class diagram. The magnified part shows a node with the class name, attributes, and functions. . . 202
7.5 The artifact that was used in task Recreate with its three ImitGraphs. 203
7.6 (a) The ImitGraph given to the participants and an enlarged region. (b) The hierarchical structure is highlighted (not shown to the participants). 205
7.7 The artifact used in the first setting of task Matching. A central node and two secondary nodes are magnified, showing the codes they contain. 206
7.8 Experience of users in using (a) graphical models and (b) tablets, (c) Experiment Setup 209
7.9 Completion times for four tasks 213

7.10 Search Task: consumed time in each state 214
7.11 Recreate task: consumed time in each state 215
7.12 Search task: the number of errors of different types 217
7.13 Hierarchy task: the number of errors of different types 218
7.14 Total number of errors for tasks Search, Hierarchy and Matching 218
7.15 Samples of search paths: (a) and (c) ZoomScroll, (b) and (d) FlexiView. The color shows the sequence and the white nodes were missed. 220
7.16 Samples of note papers used by four participants when using FlexiView and MultiFile+ZoomScroll . 221
7.17 Two partial snapshots of the artifact used in task Recreate: (a) normal and (b) with three magnets . 222
7.18 Satisfaction with ease of use of the used technique after each execution. The third column depicts the difference. 223
7.19 Final survey results: S1: I learned to use FlexiView quickly. S2: Other people will learn to use FlexiView quickly. S3: Users have the overview of the artifact more often by using FlexiView. S4: Users are more aware of their position inside the artifact by using FlexiView. S5: Users become more productive by using FlexiView. S6: Users make fewer errors by using FlexiView. S7: I recommend FlexiView to be implemented in other tools. S8: I am satisfied with the ease of using FlexiView. 224

Chapter 1

Synopsis

1.1 Introduction

Creating a wrong product is one of the causes of failure in software development [Lin99]. Lack of proper understanding of what customers require and uneven distribution of this understanding among the stakeholders result in a product that is not expected [TR04]. In contrast, modeling the requirements of a system from different points of view ensures better understanding, and detailed documentation of the models ensures homogeneous distribution of the understanding [ELC+98, BSN12]. However, modeling and documenting requirements are known as two costly activities in Requirements Engineering (RE) and are neglected to save time [dSAdO05]. In many cases, this strategy is self-defeating since poorly documented requirements result in mistakes that only can be discovered late in the project. The time and effort needed

for fixing late-discovered mistakes nullify the benefits acquired by minimizing the documentation [ZGYS+15] and, in the worst case, lead to creating a wrong product.

Adequately documented requirements are essential for understanding a system [Som01]. The amount considered as adequate may differ in different methodologies. For example, iterative software-development approaches tend to decrease the amount of documented requirements. Instead, they provide the chance of evaluating the under-development product more often [dSAdO05] in order to make early discovery and recovery of misunderstandings possible. Repeatedly creating versions of the product that can be evaluated imposes an overhead and in some cases is not possible. No matter how much documentation a methodology requires, achieving it without exceeding the allocated cost is challenging. An alternative way to face this challenge is increasing the performance of requirements documentation tasks. Improving the performance of requirements documentation tasks is always appreciated since it raises the probability of creating the anticipated quantity and quality of documented requirements without increasing the required time and effort [GGM+13].

Requirements engineers comprehend their systems of interest and document their comprehension in various artifacts such as requirements specifications, glossaries, user stories, diagrams, charts and so on [Pet09, Lis15]. The main purpose of creating artifacts is storing the comprehension of the system. The stored comprehension can be transferred for being evaluated or being used later by the creator or other stakeholders [CKI88]. Depending on the complexity of a system, different numbers of artifacts may be needed. Each

artifact captures an aspect of the system and they all together create a representation of the system suitable for the underlying purpose [AKGC14]. These artifacts are created, edited and viewed constantly by different roles [GG17a, Lis15]. Using these artifacts is not bound to the requirements engineering phase, and they are used during the whole software development process [CKI88]. A significant amount of time of different roles in a software development process involves working with requirements artifacts.

Here, we define the term *artifact* since we use it extensively throughout this dissertation:

Definition

Artifact: In this dissertation, an artifact is any type of textual and/or graphical document that is created to represent the requirements of a system or model its functionality.

Each requirements engineering team uses a set of software tools for creating, editing and viewing artifacts [dGNA+12]. The requirements engineers spend a notable amount of time working with these tools. Choosing the right tool for any type of task can dramatically affect the performance of fulfilling that task and artifact-related tasks are no exceptions. A right tool needs to have powerful functional features, and at the same time, its features should be usable. For example, an advanced search function with a poor design which requires filling multiple mandatory fields will be most probably disregarded despite its power. Usability is defined as follows [Int18]:

Definition

Usability: "The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use"

In this dissertation, our focus is on software tools that requirements engineers use for artifacts. We call them requirements engineering tools when we consider them to support all types of artifacts, and we call them requirements modeling tools when we consider them to support graphical models:

Definition

Requirements Engineering Tool: In this dissertation, requirements engineering tools are software tools that are used for creating, viewing and editing any type of requirements artifacts.
Requirements Modeling Tool: In this dissertation, requirements modeling tools are software tools that are used for creating, viewing and editing graphical models.

Since the user interface (UI) is the main communication channel between users and applications, it directly affects the usability factors, i.e., the efficiency, effectiveness and user satisfaction [MHP00]. The UI is responsible for presenting the information to the users and receiving their commands. Via commands, the users may define which part of the information should be displayed and how

the presentation should be structured. The commands also capture how the users intend to alter the information. Additionally, the UI is responsible for making the users aware of the features that are at their disposal. Various guidelines are created to help software designers to achieve high usability [FHH00, Bev95]. However, a set of guidelines that ensures usability in all types of applications does not exist. Designing a usable UI depends on the characteristics of the users, the type of data that is dealt with and the complexity of the operations that should be supported [May99]. These dependencies increase the need for tailored UI designs and usability tests for validating their usability.

In requirements engineering, the data is recorded as artifacts which can be stored either in files or repositories [GG17a]. Since the type of data affects the usability of the UI, the characteristics of requirements artifacts should be considered when designing requirements engineering tools. For example, requirements artifacts are usually large, and when displayed entirely on the screen, their details are too small to be read. In order to tackle this challenge, software tools provide zooming functions with which the users can scale up the artifact and view only a portion of it on the screen in a readable size. Then, the tools provide scrolling functions so that the users can change the focus point and navigate to the parts that are located outside of the screen. This traditional solution makes working with large artifacts possible, but introduces other challenges that may affect the usability, e.g., the time that is required for navigating between different parts of the artifact, losing the endpoints of the connections in a diagram, and tendency to memorize some parts of the artifact in order to decrease the number of

repeated back and forth navigations. The excessive time required for doing a task reduces its efficiency. At the same time, relying on the memory increases the probability of making errors which in turn decreases effectiveness and satisfaction [GG17a].

The above example shows how the common problem of navigating in large artifacts that had a well-known solution for years, e.g., zooming and scrolling, still can exert an adverse effect on the usability of the tools. Working with large artifacts occurs frequently and almost all requirements modeling tools support zooming and scrolling as a workaround [GAG17]. However, there are techniques that address this problem with different approaches. For instance, the so-called focus+context navigation techniques [CKB09] show the focused area of the artifact with high details while keeping the rest of the artifact as a surrounding context with lower details on the screen. Although this type of navigation techniques has been used in various fields and applications, its effectiveness in the field of requirements engineering is not empirically studied. Many similar problems in requirements engineering tools can be identified that have commonly-used solutions while having alternative solutions. In order to make a wise decision about whether an alternative solution should be added or replaced in requirements engineering tools, we need thorough studies in which the usability of these techniques are fairly and precisely evaluated and compared in the context of requirements artifacts.

1.2 Background and Motivation

The first graphical user interface (GUI) emerged within the 1980s, and within few years, all applications were equipped with a GUI unless a textual console was mandatory. Almost 20 years after the first GUI, Myers et al. [MHP00] predicted in 2000 that user interfaces would radically change. During the 18 years that passed since this prediction, we observed how GUIs have evolved and features that were once fancy became standards. However, no radical change happened especially in requirements modeling tools. In order to know whether this lack of change was due to the sufficiency of the available GUI features or was a negligence, two questions should be answered: 1) Are there usability challenges in current GUIs of requirements modeling tools? If the answer is positive, we should answer the second question: 2) Are there alternative user interface techniques that can potentially enhance the usability of requirements modeling tools? We performed a literature review to answer these two questions and designed the research presented in this dissertation based on these answers.

In this section, we review two types of related works: (i) The empirical studies that concern RE in general and the usability of RE modeling tools in particular. This part answers the first question. (ii) The studies that concern information presentation and navigation techniques in modeling tools and management of the level of detail. This part answers the second question. Finally, we present our motivation after answering these two questions.

1.2.1 Empirical Studies on Requirements Artifacts and Tools

Empirical studies in RE cover different topics such as the extent to which improving RE affects the whole software development process [DC06], user involvement in software engineering [PB13], and effectiveness of elicitation techniques [DDH+06]. A few empirical studies addressed requirements artifacts, requirements tools, and their usability.

Different artifacts are created in requirements engineering. Winkler [Win07] has performed a survey to find which artifacts are created in the RE phase of software engineering and how information flows between related artifacts. The information flow between artifacts makes them interconnected. By interviewing practitioners, Liskin [Lis15] studied how multiple interconnected artifacts are being used and managed. Another known characteristic of requirements artifacts is that they are mostly outdated and poorly written. This is confirmed by Lethbridge et al. [LSF03] in a research where they used interviews and surveys as data collection techniques to study how software engineers use requirements artifacts. They did not specify any reason for their finding. The known characteristics of requirements artifacts are limited to such sparse studies and accurate statistics are not available.

There are many tools developed specifically for requirements engineering. A list of such tools has been created by De Gea et al. [dGNA+11]. They have also compared the features of these

tools. In spite of the existence of many specialized RE tools, Forward et al. [FL02] reported that most practitioners preferred general-purpose tools such as word processors and text editors for creating artifacts. Karlsson et al. [KDoD+02] reported that small companies request simpler RE tools than the existing ones. Their reasons were that sophisticated RE tools require an extensive introduction and have a steep learning curve. There are studies that aimed for improving the requirements tools, e.g., the study performed by Hoffmann et al. [HKWB04] in which they created a list of requirements for requirements tools. However, it is not clear why the existing RE tools are not the first choice of requirements engineers.

Preferring general tools over RE tools and characteristics such as steep learning curve in aforementioned studies imply that there are usability challenges in RE tools. However, a few works explicitly studied the usability of RE tools. For example, De Alwis et al. [DAMR07] performed an exploratory study on three different software exploration tools. They could not find any evidence of practical benefits in using those tools. Roehm et al. [RTKM12] found that developers do not know their tools and are not aware of some the standard features that their tools provide.

By reviewing the literature on this topic, we found that there is evidence showing that the users of requirements tools face usability challenges. However, we did not find a clear picture of the existing challenges. The lack of knowledge in this area was evident. In summary, we face a knowledge gap regarding the usability challenges of requirements tools and the characteristics of artifacts that are relevant to these challenges.

1.2.2 Information Presentation and Navigation in Requirements Modeling Tools

Information is the centerpiece of requirements modeling tools. Requirements engineers store information in their textual and graphical artifacts. Requirements tools display the artifacts on the screen so that the users view the information and change it if they want. Due to having limited screen sizes, displaying the entire information stored in different artifacts on the screen is not possible. Two categories of approaches address the problem of limited screen size: increasing the screen space or utilizing the available screen space more efficiently. In the first category, large screens or arrays of multiple screens are used. Larger display screens improve the productivity by improving the performance of the users. This is reported in an empirical study by Czerwinski et al. [CSR+03]. Using more than one screen also improves the performance. But, we cannot add to the number of screens without a limit. At some point, more screens will be useless. Lischke et al. [LMW+15] found that the optimal number of screens is three. They reached this conclusion after conducting controlled experiments with one to six screens.

In the second category, different techniques are employed to use the available screen space for displaying the most relevant information on the screen. Cockburn has categorized information navigation techniques into four groups: zooming, overview+detail, focus+context and cue-based techniques [CKB09]. Zooming has been supported in RE tools for a long time and there are RE

modeling tools that have an overview+detail approach [FD10]. In overview+detail techniques, an overview of the artifact with less detail is shown in addition to the artifact. Focus+context is a general concept of having different levels of detail in single view of the information: a higher level of detail for the focused area and a lower level of detail for the context area. Fisheye technique is the theoretical foundation of the focus+context concept introduced by Furnas in 1986 [Fur86]. Graphical fisheye view is a technique that applies this concept to graphical models [SB94]. It allocates more space to the focused area so that more details can be displayed and less space to the context area so that the context information can be displayed with coarse details around the focused area. The fisheye concept is used in the technique proposed for visualizing ADORA models [GBJ02,RMS+08,RMG07]. In this technique, users can enlarge a part of a diagram without enlarging the other parts. The space needed to fit in the enlarged object is provided by pushing other objects away from the enlarged object. Cue-based techniques facilitate navigation in large information spaces by providing additional information, e.g., highlighting a certain type of information so that it can be found easily in a dense information visualization or displaying proxies at the edge of the screens for the elements that are located out of the visible area [FD10].

None of the mentioned techniques can completely solve the challenges of navigation in large graphical models [RCB+05]. Different graphical models have their own characteristics and require a navigation technique tailored to their needs. For example, the visualization of artifacts in automatic tracing tools can benefit from hierarchical structures according to Cleland-Huang et al. [CHH07].

In another example, Reinhard et al. [RMG07, RMS+08] developed a custom-made presentation technique based on the fisheye concept to enhance the experience of the users who work with their requirements modeling language ADORA.

Our literature review revealed that dedicated information presentation and navigation techniques in RE are designed for very specific use cases. For example, the ADORA visualization technique is designed to work with the ADORA modeling language and the off-screen technique developed by Frisch et al. [FD13] works with class diagrams. Therefore, the applicability of such techniques is limited. Although they can be potentially generalized, we did not find a navigation technique which is tailored to RE needs and can be applied to a wide range of requirements artifacts and tools.

1.2.3 Level of Detail, Semantic Zooming, and Cognitive Load

The information density varies in different artifacts and it depends on the decision of requirements engineers whether they want few dense artifacts or many light artifacts [FvH10]. Regardless of this decision, the user of an artifact does not need the entire information embedded in the artifact at all times. Sometimes higher level information suffices for a task. Hiding the unnecessary information is one way of increasing the efficiency of screen space usage [FB95]. Semantic zooming is the concept of changing the level of detail

based on the demand. For example, the amount of information can be adjusted to the zoom level in zoomable user interfaces [BH94]. There are different techniques for changing the level of detail. In a hierarchical structure, an entity can be replaced by its underlying entities when the zoom level increases, which results in a higher level of detail [FDB08]. Onion graphs is a visualization technique for UML graphical models that combines the concept of focus+context with semantic zooming [KM07].

In addition to being used for saving space, semantic zooming is also used for reducing the cognitive load [Par13b]. Presenting information hierarchically helps the human mind in understanding it. If the information is presented at once, the mind tries to derive a hierarchical structure itself [KDvV02]. That is why in an information-intensive field such as requirements engineering, the artifacts are created with several abstraction levels. For example, Cornelissen et al. [CVDMZ07] defined a set of metrics for scenario diagrams which can be used for recommending the number of abstraction levels that result in the optimum amount of detail in each level.

Requirements engineers maintain mental images of artifacts and the relationship between artifacts which is another factor that affects the cognitive load [Pet09]. A mental image of an artifact helps its owner in understanding the artifact, finding a piece of information, planning the current task and in general, reduces the amount of navigation within or between artifacts. Similar to the abstraction layers, if the mental image is not supported externally, e.g., by the tool, the requirements engineers have to invest more

effort to maintain it manually. The creation and maintenance of mental images can be facilitated by the visualization and navigation techniques. For example, reducing the level of details prepares the information for being captured as an image, and displaying an overview of an artifact provides such an image directly although it has to be adapted by the users based on their needs.

Adaptive level of detail, semantic zooming and supporting the mental image of the artifacts can potentially lower the cognitive load of working with requirements artifacts. Therefore, they affect the usability of RE modeling tools. Nevertheless, we did not find a published study discussing the effectiveness of these concepts in RE modeling tools.

1.2.4 Distortion and Physical Metaphors

Adjusting the visual presentation of an artifact based on the user's current needs involves repositioning of elements and non-uniform transformations such as scaling and changing the level of details [LA94]. Transforming different parts of an artifact non-uniformly causes an adverse effect known as distortion. Although distortion is inevitable in this type of approaches, the amount of distortion depends on the strategy of the approach for putting back together the transformed parts. For example, in ADORA visualization [GBJ02, RMS+08, RMG07], the distortion is minimal since the relative positions of the elements are preserved. However, unwanted empty spaces may appear between elements after transformations. A more compact layout can be achieved at the cost

of creating more distortion which in turn causes loss of focus and disorientation. To alleviate the resulting disorientation, we can take measures such as using physical metaphors [RK14].

Physical metaphors aid virtual environments to look natural to the users [RK14]. Graph visualization techniques have used metaphors such as magnets, springs, and forces to make their results more understandable and consequently more acceptable. A force-directed graph visualization is an approach for optimizing the layouts of graphs. This approach arranges the nodes of a given graph in an aesthetically pleasing way by improving metrics such as the number of crossing edges and lengths of edges. Their mechanism is modeling the nodes and edges as magnetic elements and simulating the resulting structure. The elements move because of the forces that they apply to each other [SF08], which eventually leads to an optimized layout. One of these approaches named spring model [Ead84] uses metaphors of magnets and springs to model nodes and edges of graphs respectively. Then, by exposing the resulting model to a magnetic field, the layout is optimized. In addition to layout optimization, physical metaphors are used for search and filtering purposes [RK14].

1.2.5 Motivation

General-purpose tools such as word editors and diagram tools have been used widely for a broad range of purposes. Their main goal is to satisfy various types of users. Therefore, they prefer to incorporate features that will be used by a larger number of users.

In the end, such tools lack dedicated features for specific users and provide features that are useless to a part of users. Additionally, they can barely make assumptions about how their users will use their tool. Consider the following example. In a general-purpose tool, for designing a class diagram, the designer should create boxes with three separate sections manually while a dedicated tool can create such boxes with single actions. Furthermore, settings such as colors, shadows, and fonts are mostly irrelevant for designing standard class diagrams. As a result, the usability criteria of general-purpose tools differ from the usability criteria of dedicated tools such as RE tools. The literature about the usability of RE tools is sparse and limited. Therefore, we can hardly answer the first questions asked at the beginning of this section. However, as discussed in 1.2.1, there are hints in the literature suggesting that the users of RE tools face usability challenges, e.g., preferring general-purpose tools over RE tools and not knowing the features of the RE tools that are used. A decisive answer to this question requires further investigation. Thus, we dedicated a part of this research to discover the challenges of using RE tools and the influential characteristics of artifacts, screens, and tools.

Regarding the second question, we found four points in the literature: (i) There are paradigms, e.g., focus+context, overview+detail and cue-based techniques, that enhance the navigation within the information visualizations and can be effective in RE. (ii) There are also techniques, such as semantic zooming and physical metaphors, that can be combined with other navigation techniques to improve the user experience of working with graphical models. (iii) Screen space is an important factor since several studies focused on the

practitioners' screen size and the number of screens used by them when working with large models or multiple models at the same time. (iv) Usability tests and comparisons on RE tools are rarely done because of the high cost of conducting such experiments with various modeling languages used in RE.

All these findings from our literature review convinced us that tailoring a navigation technique specifically for RE modeling tools can improve the usability of these tools. Since requirements engineers work often with RE models, an improvement in the usability of RE tools can positively influence their overall performance. Therefore, we frame our thesis statement as follows:

Thesis Statement

Requirements engineers invest a significant amount of their time in working with requirements modeling tools to create, modify or understand artifacts. Therefore, improving the usability of these tools by employing tailored navigation techniques can enhance the users' performance when working on requirements engineering tasks.

1.3 Research Goal, Questions and Methodology

In this section, we introduce our research goal, derive research questions from our goal, and present an overview of our research methodology.

As described in Section 1.2.1, the usability challenges in RE modeling tools are not completely known. A part of our goal is to fill this knowledge gap. After knowing the existing challenges we devise a dedicated information presentation and navigation technique for RE artifacts and evaluate its effectiveness. Therefore, we establish our goal as follows:

Thesis Goal

Investigate the usability challenges in working with RE artifacts, create a new navigation technique that addresses the challenges found, and experimentally compare the new technique with a commonly used technique to evaluate its effectiveness.

The research method of this thesis is inspired by the conceptual framework of empirical research proposed by Wieringa and Heerkens [WMMR05] at the highest level. We follow the six engineering steps: Problem investigation, solution design, design validation, choose a solution, implement the chosen solution, and implementation evaluation. However, this research comprises three major phases with different research characteristics at a finer level: problem, solution, and evaluation. In the first phase, we search for the unknown challenges of working with RE artifacts. This phase concerns a knowledge problem, and an exploratory research method is suitable for it. We use interviews and surveys as data collection methods to gather statistics about requirements artifacts, tools, and the challenges of working with them. In the second

phase, we develop new concepts by combining and extending existing concepts. Two new concepts are devised in this part: a new navigation technique for requirements artifacts and a new graphical notation for being used in usability tests. The problem addressed in this phase is a world problem and a constructive research method is appropriate for it. In the third part, we conduct controlled experiments to evaluate the solution that we have designed in the second phase. Since we need to gather precise data during the experiments, we select instrumentation as our data collection method. This problem is again a knowledge problem and requires an empirical research method.

To achieve our goal we formulated four research questions. Figure 1.1 shows the four research questions along with the three phases of this research (problem, solution, and evaluation), the engineering steps, the problem type (world or knowledge), and the research method of each phase (exploratory, constructive, or empirical).

We framed our first research question to fill the knowledge gap of the usability challenges of navigation within artifacts and the characteristics of artifacts that are relevant to these challenges. This question belongs to the problem phase and is a knowledge problem. We can categorize it as an existential question according to Easterbrook et al. [ESSD08]:

Research Question 1

What challenges do practitioners face when navigating inside requirements artifacts and which characteristics of artifacts do influence these challenges?

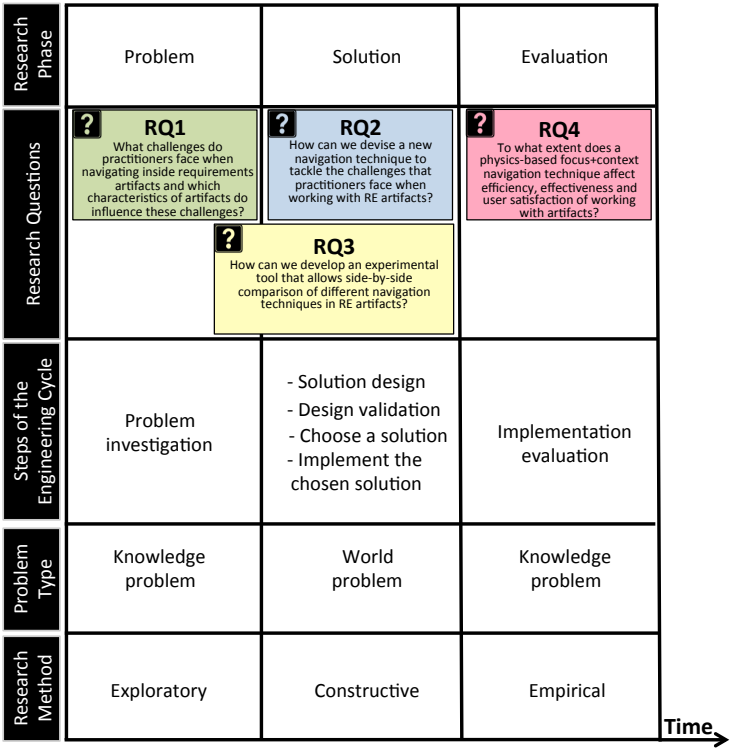


Figure 1.1: The phases of this research, their corresponding research questions, the steps of the engineering cycle, the problem type and the chosen research method of each phase

As the main goal of this study is about tackling usability challenges of working with RE artifacts, we should identify these challenges in the first step. We should search for the usability challenges that practitioners face in the real world. Additionally, we should find the cause of the challenges in order to pursue solutions for them. Therefore, as we investigate the potential challenges, we also seek for the relevant properties of the artifacts that may cause such challenges. We cannot exactly predict what challenges we may encounter or what properties of the artifacts we should measure. We use the knowledge we gained from the literature review as a guide but we make the investigation open to new challenges and properties. For example, we understood that screen space is limited, thus the size of artifacts is relevant. Also, we knew that many practitioners prefer general-purpose tools, thus we can ask about their reasons in order to identify the challenges.

As Figure 1.1 shows, the second research question belongs to the solution phase of this research and is a world problem:

Research Question 2

How can we devise a new navigation technique to tackle the usability challenges that practitioners face when working with RE artifacts?

In the initial literature review, we encountered information navigation concepts that could potentially solve the challenges of working with RE artifacts, e.g., focus+context navigation techniques, Physics metaphors, and semantic zooming. We came up

with this question: How can we combine these concepts in order to create a new navigation technique for RE artifacts? Although this question can be approached in different ways, we focused on these concepts because focus+context techniques were proven to be useful for navigating in large models. However, distortion was a known side effect of these techniques. On the other hand, the physics metaphors were successful to make complex interfaces more natural to the users and could potentially alleviate the distortion effect of the focus+context techniques.

A part of the third research question belongs to the problem phase and is exploratory. The larger part of it belongs to the solution phase and concerns a world problem (Figure 1.1):

Research Question 3

How can we develop an experimental tool that allows side-by-side comparison of different navigation techniques in RE artifacts?

From the literature, we knew that usability tests are costly and we had to test the outcome of research question 2. This motivated us to create a framework with the following properties: it allows (i) precise and fair comparison of navigation techniques in a RE context, and (ii) new navigation techniques to be implemented quickly. In order to create such an experimental tool, we had to investigate the existing requirements tools to find their common features. This investigation made the exploratory part of this research question. In order to have generalizable results when

using our experimental tool, it should support as many modeling languages as possible. However, developing such an experimental tool is time-consuming and implementing a navigation technique for a wide range of modeling languages is costly. Therefore, we were motivated to find a solution for producing generalizable results in usability tests of navigation techniques without having to support a wide range of modeling languages.

Our fourth research question belongs to the evaluation phase of this research and is a knowledge problem. We can categorize this question as a causality-comparative interaction question according to Easterbrook et al. [ESSD08]:

Research Question 4

To what extent does a physics-based focus+context navigation technique affect efficiency, effectiveness and user satisfaction of working with artifacts?

Finally, we want to evaluate the technique that we devised in response to research question 2 using the framework that we developed in response to research question 3. This research question is an experimental question. To answer this question we conduct controlled experiments in which we compare our navigation technique with a traditional technique. We gather user interaction data during the experiments. By analyzing the gathered data we can determine how much the performance of the users changed and to what extent the challenges that we found in research question 1 are addressed. We measure three factors of usability for this purpose: efficiency, effectiveness, and satisfaction.

1.4 An Overview of FlexiView

In this section, we explain FlexiView, our navigation solution for RE artifacts. We have devised this technique mainly to answer the second research question, and later, we augmented it with more practical details when implementing our experimental tool to answer the third research question. Therefore, the detailed description of FlexiView can be found in Chapters 3 and 6.

FlexiView is an information presentation and navigation technique designed specifically for RE artifacts. It incorporates three paradigms: focus+context, physics metaphors, and semantic zooming. The main goal of FlexiView is to display a large artifact on a limited screen space. FlexiView adjusts the presentation of the artifact based on the users' demand. As a focus+context technique, it allocates the screen space to different parts of the artifact heterogeneously according to the importance of each part indicated by the user. The user indicates importance by placing virtual magnets, and FlexiView allocates the screen space using a physics simulation. For this simulation, FlexiView partitions the screen space first. The partitioning structure is a graph. The vertices of this graph are the corners of the screen and the virtual magnets together. The edges partition the screen space into several triangles that cover the whole screen space. Then, FlexiView transforms the edges and vertices of the partitioning structure into a structure of virtual springs and magnetic balls. The simulated interaction between the virtual magnets, springs, and balls results in a transformation of the shape and size of the partitions, which

are proportional to the magnetic powers specified by the user. FlexiView also adjusts the amount of detail of each element of the artifact according to the actual size of the element on the screen, which is known as semantic zooming.

We explain the working mechanism of FlexiView by going through an example. Figure 1.2a shows a simple graph. In Figure 1.2b, the user expresses his/her interest by putting a virtual magnet on the upper-left node. Behind the scene, FlexiView partitions the space using Delaunay triangulation, and models the edges and vertices of the triangulation by virtual springs and magnetic balls (all with the same polarity as the virtual magnet). In this state, the created virtual physics environment is in balance and the forces of springs, balls, and the magnet neutralized each other. When the user increases the magnet strength in Figure 1.2c, the structure is not balanced anymore. The magnet and other magnetic balls apply more forces to each other that cannot be neutralized by the forces of the springs anymore. Therefore, magnetic balls move until the whole system gains balance again. The distortion caused by the change in the shape of the regions provides more space for the magnetized node. In Figure 1.2d, the user puts another virtual magnet on the lower-right node, and FlexiView re-partitions the space based on both magnets. When the user increases the power of the second magnet in Figure 1.2e, the forces applied to other balls cause a change in the structure and consequently provide more space for the node carrying the new virtual magnet. The whole virtual structure consisting of the springs and balls is not visible to the user. Figure 1.2f shows how the user actually sees the graph. The magnetized nodes are enlarged, and the other nodes

are shrunk. This allows the user to see more detail of the enlarged nodes while their connections to other nodes are still visible. More details about how FlexiView works can be found in Chapters 3 and 6.

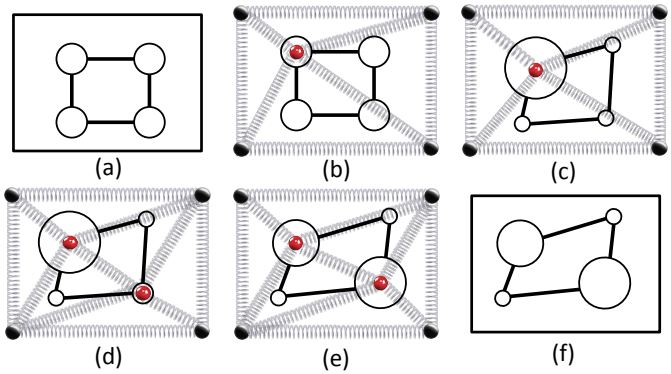


Figure 1.2: The working mechanism of FlexiView: (a) A given graph. (b) A virtual magnet is put on the upper-left node. (c) the strength of the magnet is increased. (d) A second magnet is put on the lower-right node. (e) The strength of the second magnet is increased. (f) How the user actually sees the graph.

In the following scenario, we explain how FlexiView can be applied to requirements modeling tools. In this scenario, we compare FlexiView and zooming when a user searches in a large model. Figure 1.3a shows a graph that can be a simplified representation of a class diagram or a goal decomposition model. The node that the user is interested in for starting the search is indicated by a red arrow. The details of the nodes are not visible at this scale, and only the first letter of the content of each node is shown.

In Figure 1.3b, the user zooms in on the node of interest. As a result, only the node of interest is visible on the screen while all other nodes are located outside of the screen. The connections are partially visible such that the other ends of the connections are unknown. In Figure 1.3c, the user magnetizes the node of interest. FlexiView partitions the screen space behind the scene and distorts the graph such that more space is allocated to the node of interest. Therefore, the node of interest is enlarged and its details are visible while the rest of the nodes are shrunk.

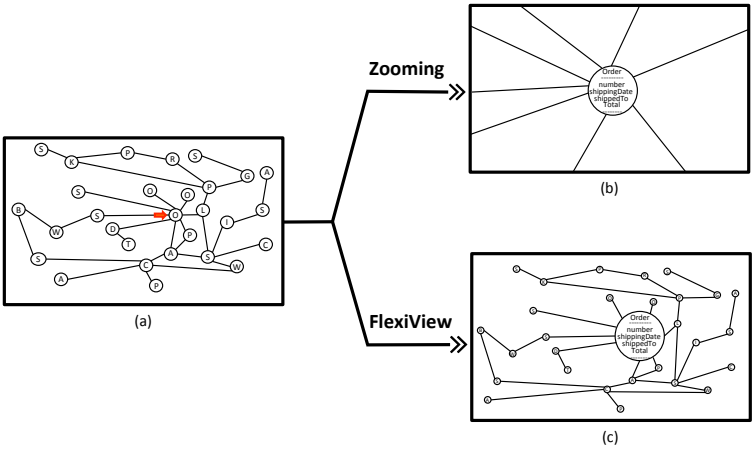


Figure 1.3: A comparision between zooming and FlexiView. (a) A given graph in which the node of interest is indicated with a red arrow. (b) The node of interest is zoomed in. (c) The node of interest is magnetized using FlexiView.

In both Figures 1.3b and 1.3c, the user can check the information inside the node of interest. The difference is that the user does not see the neighboring nodes when zooming but sees all the other

nodes when using FlexiView. Having all the nodes on the screen, the user can immediately decide which node to check next when checking the node of interest is finished. When zooming, the user has two options for deciding which node to visit next: either to zoom out to see an overview of the model or rely on his/her memory and scroll toward the direction of the next node without zooming out. The first option requires more time and the second option is error-prone. Detailed results of experiments comparing these two techniques can be found in Chapter 7.

1.5 An Overview of ImitGraphs

Various modeling languages with different graphical notations are used by requirements engineers. The characteristics of these notations are different and as a result, they behave differently under editing operations. For example, moving an element may be possible individually in a model, may affect the neighboring elements in another model, or may be even impossible. The behavior of graphical models is described in Section 4.3. Due to the variety of behaviors in graphical models, the usability challenges are different and the solutions for the challenges are different, too. Therefore, software designers should take different graphical notations into account when researching the usability of requirements modeling tools. This makes usability testing and usability experiments time-consuming and cumbersome.

The idea behind ImitGraphs was to create a single graphical notation which can represent a variety of concrete graphical notations

and to use this single notation for usability testing and experimentation. To achieve this, the intended notation should imitate the behavior of the notations that it represents. We based our new notation, which we call ImitGraphs, on simple node-and-edge graphs since these are the common ancestors of most of the graphical notations. Then, we added additional properties such that, without changing their simple appearance, ImitGraphs can be customized to imitate the behavior of other graphical notations. Figure 1.4 shows three parts of different graphical models and their equivalent ImitGraph: (a) activity diagram, (b) class diagram, and (c) goal model. We continue by describing the components of the ImitGraphs.

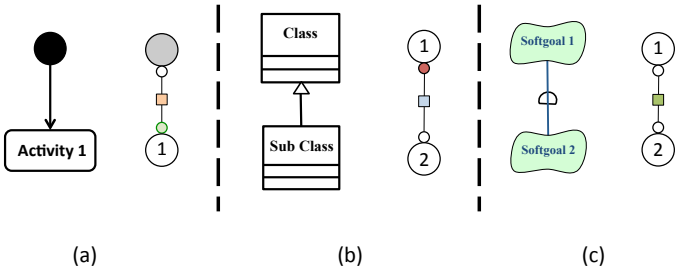


Figure 1.4: Parts of three different graphical models and their equivalent ImitGraph: (a) activity diagram, (b) class diagram, and (c) goal model

1.5.1 ImitGraph Components

ImitGraphs are composed of three types of components: joints, connections, and nodes. Figure 1.5 shows a simple activity diagram,

its equivalent ImitGraph, and pairs of corresponding components of both diagrams.

Joints. The connections in different graphical notations have different endpoints. In Figure 1.4, different types of endpoints in three different graphical notations can be identified, e.g., straight, arrow, and triangle. In ImitGraphs, a joint is a circular element at each end of a connection. Each joint has two properties: (i) its color that indicates its type and (ii) its ability to hold a text. In Figure 1.4, straight endpoints are represented by white joints, the arrow endpoint is represented by a green joint and the triangle endpoint is represented by a red joint.

Connections. In ImitGraphs, a connection consists of two joints, a line that connects these joints, and a square in the middle of the line. A connection has the following properties: (i) the color of the center square, which indicates the type of the connection, (ii) its ability to hold a text, (iii) the types of its two joints, and (iv) its orientation, which can be horizontal, vertical or any.

Nodes. ImitGraph nodes are circles larger than joints. A node has the following properties: (i) its color, which indicates its type, (ii) its ability to hold a text, (iii) the connection types that are allowed to be attached to it, (iv) the joint type of each connection type that is allowed to be attached to it, (v) the maximum number of each connection type that can be attached to it, and (vi) the anchor point of the node that each connection can be attached to. Anchor points can be top, bottom, left, right, any, or a combination of these.

1.5.2 ImitGraph Commands

ImitGraphs do not have meaning, unlike the graphical models that they represent. Even though this property enables ImitGraphs to represent different graphical models, it makes designing tasks for usability tests challenging. To tackle this challenge, we introduced ImitGraph commands. The commands are partially textual and partially visual and can be used to instruct the participants of usability tests to make new models or modify an existing model. The commands are explained in Chapter 4.

1.5.3 An Example Scenario

We show how ImitGraphs can be used in an experiment by going through a simple example scenario. In this scenario, we first define an ImitGraph that imitates a simple activity diagram. Then, we show how commands can instruct a participant to create an ImitGraph that corresponds to the simple activity diagram. The participants are not aware of the relation between the diagram they create and the activity diagram. A more advanced scenario is described in Chapter 4.

Figure1.5 shows the activity diagram that we want to be imitated by an ImitGraph. It has three types of nodes, two types of joints and one type of connection.

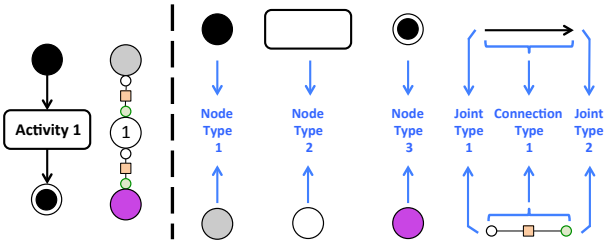


Figure 1.5: An activity diagram, its corresponding ImitGraph, and pair-wise corresponding constituents of them.

We start by defining the joints. Table 1.1 contains the definitions of the two joints: J1 represents the straight endpoints of the connections, and J2 represents the arrow endpoints of the connections. None of them holds a text.

Table 1.1: Joint Types

Type	Symbol	Label
J1	○	No
J2	○	No

Table 1.2 contains the definition of the only connection type we have. Its color is beige and can not have a text. At one end it has a joint of type J1 (straight) and at the other end, it has a joint type of J2 (arrow). Its orientation is restricted to be vertical.

Table 1.2: Connection Types

Type	Symbol	Label	First joint	Second joint	Orientation
C1	□	No	J1	J2	Vertical

Table 1.3 contains the definitions of the three node types: N1 is gray and represents the start node of the activity diagram. It does

not hold a text. Maximum one connection of type C1 from the J1 endpoint can be connected to the bottom of this node type. N2 is white and holds a text since it represents the activity nodes in the activity diagram. Two connections of type C1 can be connected to this node type; one from J2 endpoint to its top and one from J1 endpoint to its bottom. N3 represents the final node in the activity diagram. It is purple and does not hold a text. Only one connection of type C1 from its J2 endpoint can be connected to the top of this type of node.

Table 1.3: Node Types

Type	Symbol	Label	Connection type	Joint type	Max	Connection point
N1		no	C1	J1	1	bottom
N2		yes	C1	J2	1	top
			C1	J1	1	bottom
N3		no	C1	J2	1	top

After defining the properties of the components of the ImitGraphs, we can use them in a usability test. In a usability test, we give the participant a task to perform. The first two columns of Figure 1.6 show three steps of a task when using the activity diagram notation. The first column contains the textual description that corresponds to each step and the second column contains the resulting diagram. The last two columns of Figure 1.6 show three steps of a similar task when using the ImitGraphs notation. In the first command, the participant is instructed to create a gray node. The second command instructs the participants to attach a connection to the gray node he/she has created in the previous command. Then,

put a white node containing the letter “1” at the other end of this connection. The result is illustrated in the second column. In the third row, the participant is instructed to attach a new connection to the white node he/she created in the previous command and put a purple node at the other end.

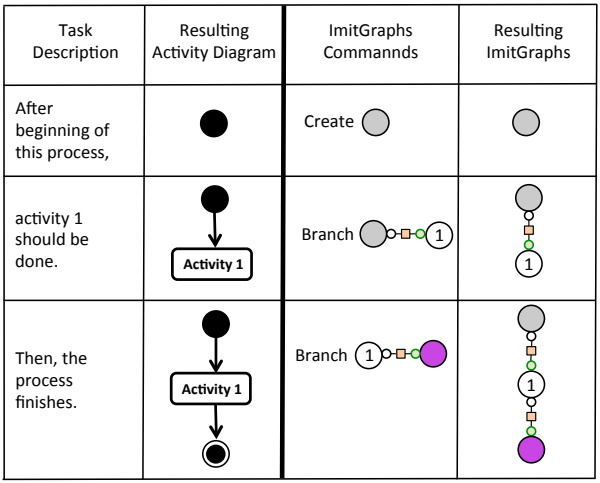


Figure 1.6: Three steps of two equal tasks in the activity diagram notation and the ImitGraph notation.

1.6 Contributions

In this thesis, we have six contributions: (i) the characteristics of artifacts and the challenges of working with them, (ii) the FlexiView technique, (iii) the ImitGraph graphical notation, (iv) common

features of touch-screen requirements modeling tools, (v) the FlexiView experimental tool, and (vi) the FlexiView evaluation results. Figure 1.7 shows the relations between these contributions and to which research question they correspond.

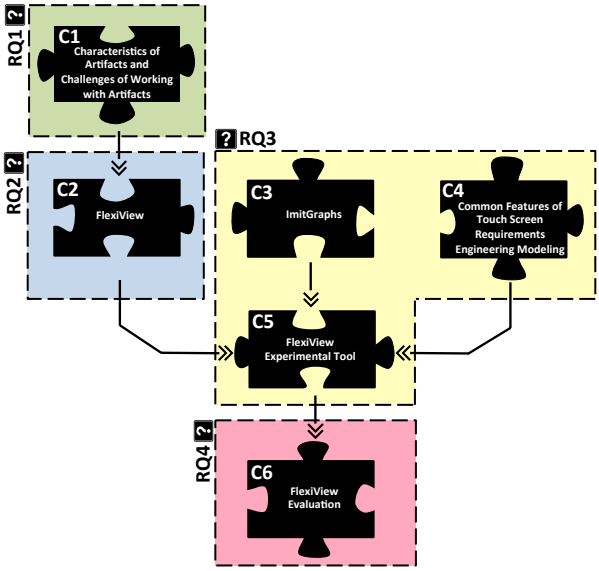


Figure 1.7: The relations between contributions and the research questions they correspond to.

C1) Characteristics of Artifacts and the Challenges of Working With Them. In order to answer the first research question, we conducted an exploratory study. The outcome of this study was twofold. Firstly, we collected information about the size of artifacts, the size of screens, the number of screens, the number of artifacts, and the number of simultaneously used

artifacts in real-world software projects. Secondly, we discovered the challenges that practitioners face when working with large and interconnected artifacts. Additionally, we prioritized the challenges. The discovered characteristics and challenges together provide a starting point for planning a research for solving the challenges.

C2) FlexiView Navigation Technique. When working with requirements artifacts, it is important to present the embedded information to the users, allow them to navigate through the information and provide them with tools to add new information and alter the existing information. To tackle the identified usability challenges in these areas, we devised a conceptual solution named FlexiView. Later, we made it a practical solution when implementing the FlexiView experimental tool. FlexiView is a physics-based focus+context navigation technique created for requirements artifacts. It has been created to enhance the performance of requirements modeling tasks by using the screen space efficiently and managing the level of detail when displaying requirements artifacts.

C3) ImitGraph Graphical Modeling Notation. Requirements engineers use different modeling languages and notations. We introduced ImitGraphs to replace these notations in usability tests. This makes usability tests of requirements modeling tools faster and more convenient. ImitGraphs are extended versions of simple node-and-edge graphs with additional properties that enable them to imitate other graphical notations. We provided a description of different components of ImitGraphs with their

properties that should be defined before use. We also provided a set of commands that can be used for instructing the participants to perform their tasks in usability tests.

C4) Common Features of Touch-Screen Requirements Modeling Tools. An experimental tool for testing navigation techniques should well represent the tools that the navigation techniques are designed for. We created a list of common user-interface features of touch-screen requirements modeling tools and the mechanisms with which they work. This list can be used to create a representative experimental tool that produces more generalizable results in usability tests.

C5) FlexiView Experimental Tool. As illustrated in Figure 1.7, we used the three contributions C2, C3, and C4 to develop an experimental tool for the purpose of usability tests. During the implementation, we provided more practical details for FlexiView. Additionally, our tool has two features that make data collections and usability test supervisions more convenient: the capabilities of (i) logging user interactions precisely and (ii) creating tasks for usability tests.

C6) FlexiView Evaluation Results. Our final contribution was made possible by all the other contributions C1—C5. They allowed us to conduct experiments in which we compared the interactions of users when using FlexiView to their interactions when using zooming+scrolling. After analyzing the gathered data, we made conclusions in the following areas: (i) how much time users spend for planning, navigating, and processing during a

modeling task; (ii) the pattern of search paths; and (iii) the errors made by the participants, the nodes that they missed, and the nodes that they redundantly visited.

1.7 Road Map and Chapter Summary

This dissertation is cumulative and comprises a collection of scientific publications. Chapters 2–7 present these publications. Figure 1.8 shows the chapters, the contributions each chapter is related to and the research questions each contribution corresponds to. In Chapter 8, a summary and the conclusion of this thesis are presented. This section presents a roadmap of this dissertation and summaries of Chapters 2–7.

Chapter 2: Visual Characteristics of Requirements Artifacts and the Challenges of Working With Them

Motivation. In our literature study, we did not find information about the usability of RE tools. However, we encountered signs of the existence of usability challenges in these tools, for example, requirements engineers preferring general-purpose tools over dedicated RE tools or not being aware of the features of the tools they use. This motivated us to perform an exploratory study to investigate how challenging RE tools are to use from the usability

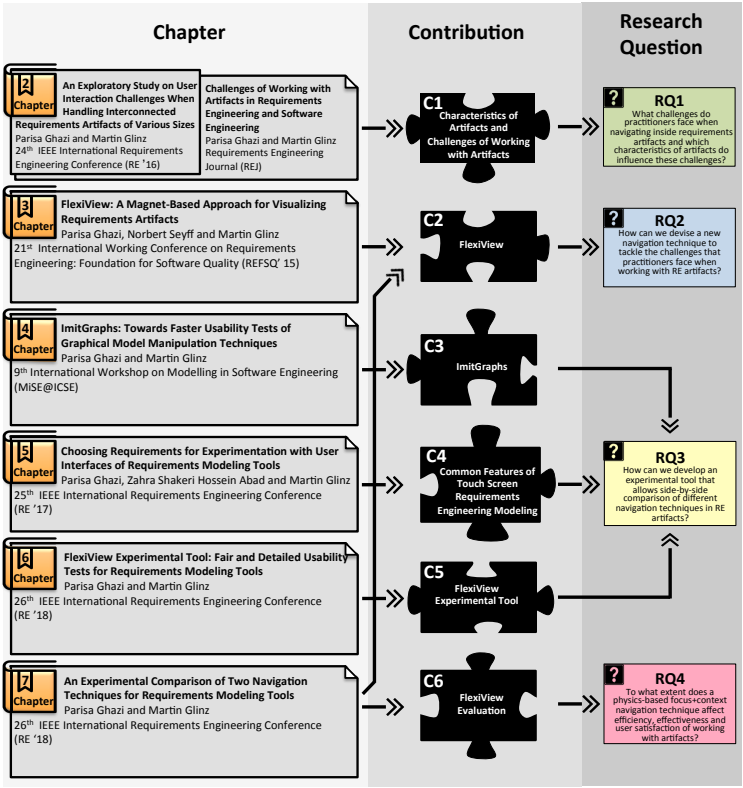


Figure 1.8: The relations between chapters, contributions and research questions.

point of view, and to identify such challenges if they exist. Additionally, we wanted to gather statistics about the characteristics of the artifacts and tools in order to trace the causes of the challenges later.

Contribution. After analyzing the data gathered in 29 interviews

and by 42 survey responses, we contributed statistics and insight about the following issues: (i) the size and number of screens being used; (ii) the size and number of artifacts, their connectivity, and their frequency of use; (iii) the challenges that practitioners face when working with large artifacts and interconnected artifacts; (iv) the workarounds that practitioners use to overcome challenges; and (v) the methods that tools use for keeping the interrelationship information of artifacts.

Research Question. In this chapter, we answer the first research question.

Chapter 3: A Physics-Based Focus+Context Presentation and Navigation Technique for Requirements Artifacts

Motivation. In the exploratory study which is presented in Chapter 2, we found that most of the artifacts do not fit entirely on the practitioners' screens and working with large and interconnected artifacts is challenging. From the literature, we knew that focus+context techniques had been used for managing large information on limited screens. However, they had the disadvantage of large drawing canvases after multiple zooming-in operations. Furthermore, physics metaphors were used to create natural user interfaces and could potentially alleviate the disadvantage of focus+context techniques. These motivated us to create a physics-based focus+context technique specifically for working with requirements artifacts on a fixed-size canvas.

Contribution. In this chapter, we present FlexiView, our solution for presenting and navigating large requirements artifact.

Research Question. FlexiView is our answer to the second research question. In this chapter, FlexiView is introduced in its conceptual state. Later in Chapter 7, we update FlexiView by adding practical and implementation details. Therefore, Chapters 3 and 7 together answer the second research question.

Chapter 4: An Imitating Graph for Faster Usability Tests of Requirements Modeling Tools

Motivation. Conducting usability tests on requirements modeling tools is costly due to the variety of graphical modeling languages used in RE. When a higher number of modeling languages are used in the tests, the results are more generalizable, but at the same time, the cost of testing increases. This motivated us to create a graphical modeling notation for usability tests that can imitate the behavior of other graphical notations. By using such a model, we can make usability tests faster while preserving the generalizability of the results.

Contribution. In this chapter, we introduce a novel graphical notation which can be customized to imitate one or more graphical notations. In addition, we present a set of commands with which we can instruct the participants of the usability test to perform tasks.

Research Question. ImitGraphs can be used to test the performance of navigation techniques such as FlexiView. Therefore, this chapter partially answers the research question 3.

Chapter 5: Common User-Interface Features of Requirements Modeling Tools

Motivation. The user interface of a requirements modeling tool has many features. These features vary among different tools. For example, each touch-screen tool has its own input mechanism for scrolling. They may use one-finger touch, two-finger touch, or other mechanisms for scrolling. When creating an experimental tool for testing a navigation technique, we need to keep the other influential parameters as similar as possible to existing modeling tools. This motivated us to conduct a market study in which we examined ten touch-screen requirements modeling tools to find the most common features of such tools.

Contribution. This chapter presents a list of features that most touch-screen requirements modeling tools support and the most common mechanism for implementing each feature.

Research Question. The contribution of this chapter helps us to increase the generalizability of the results that our experimental tool produces. Therefore, it partially answers the research question 3.

Chapter 6: FlexiView Experimental Tool

Motivation. For conducting fair comparisons between FlexiView and zooming+scrolling, we require a dedicated tool that supports both FlexiView and zooming+scrolling and is capable of recording user interactions precisely.

Contribution. In this chapter, we present how we integrated FlexiView and ImitGraphs into an experimental tool that supports the most common features of a requirements modeling tool.

Research Question. In this chapter, we provide the last piece of the answer to research question 3. Therefore, Chapters 4, 5, and 6 together answer research question 3.

Chapter 7: An Experimental Comparison of Two Navigation Techniques for Requirements Modeling Tools

Motivation. We found the challenges of working with RE artifacts (Chapter 2), devised FlexiView as a solution for a subset of these challenges (Chapter 3), and created a tool for fair and precise comparisons between our solution and existing common solutions (Chapter 6). We wanted to compare FlexiView to classic methods of navigation in large artifacts such as zooming and scrolling to investigate the influence of FlexiView on the identified usability challenges.

Contribution. For the final evaluation, we conducted controlled experiments in which the participants were asked to perform four tasks in the FlexiView experimental tool. We gathered data for 24 participants. By analyzing these data, we compared completion times, numbers of errors, and search paths. We measured two types of errors: the errors in the participants' final results and the navigation errors such as missed nodes and redundant visits. Additionally, we measured the time each participant spent on planning, navigation, and processing. Our analysis showed that FlexiView outperformed zoom+scroll with respect to task completion time, number of mistakes, cognitive load, and user satisfaction.

Research Question. This chapter mainly answers the final research question. Furthermore, it provides more technical details about FlexiView, which is a part of the answer to research question 2.

Chapter 2

Visual Characteristics of Requirements Artifacts and the Challenges of Working With Them

Original publications:

An Exploratory Study on User Interaction Challenges When Handling Interconnected Requirements Artifacts of Various Sizes
P. Ghazi and M. Glinz
International Requirements Engineering Conference 2016

Challenges of Working with Artifacts in Requirements Engineering and Software Engineering
P. Ghazi and M. Glinz
Requirements Engineering Journal 2017

Abstract

When developing or evolving software systems of non-trivial size, having the requirements properly documented is a crucial success factor. The time and effort required for creating and maintaining non-code artifacts are significantly influenced by the tools with which practitioners view, navigate and edit these artifacts. This is not only true for requirements, but for any artifacts used when developing or evolving systems. However, there is not much evidence about how practitioners actually work with artifacts and how well software tools support them. Therefore, we conducted an exploratory study based on 29 interviews with software practitioners to understand the current practice of presenting and manipulating artifacts in tools, how practitioners deal with the challenges encountered, and how these challenges affect the usability of the tools used. We found that practitioners typically work with several interrelated artifacts concurrently, less than half of these artifacts can be displayed entirely on a large screen, the artifact interrelationship information is often missing, and the practitioners work collaboratively on artifacts without sufficient support. We identify the existing challenges of working with artifacts and discuss existing solutions proposed addressing them. Our results contribute to the body of knowledge about how practitioners work with artifacts when developing or evolving software, the challenges they are faced with, and the attempts to address these challenges.

2.1 Introduction

Adequate documentation is essential for successful software development [Som01]. This is particularly true for requirements, where missing or deficient documentation may lead to developing the wrong product. As documentation consumes time and effort, it is often neglected, leading to documents which are poor or outdated [dSAdO05], [BAL15]. However, minimizing the effort for documentation does not necessarily result in time savings and less effort, since proper documentation reduces the duration of tasks, while inadequate documentation increases the risk of making mistakes and causes late discovery of the mistakes which in turn lead to a large amount of rework [ZGYS+15].

In Requirements Engineering (RE), documenting requirements is of crucial importance, regardless of the RE process or methods being used [Poh10]. Requirements can be documented in a variety of artifacts, from comprehensive requirements specifications to user stories, use cases, glossaries or diagrams. If multiple items (textual requirements, user stories, diagrams, etc.) are stored together in one document, we consider this as one single artifact.

Requirements engineers typically use tools for manipulating (creating, editing or viewing) artifacts, which range from general-purpose tools, such as text or diagram editors, to tools specifically built for RE purposes [dGNA+12]. The usability of such tools, in particular, the effectiveness, efficiency, and satisfaction with which requirements engineers can use them for viewing and editing artifacts has

a strong influence on their productivity and their willingness to produce adequate requirements artifacts.

In their daily work with requirements artifacts, requirements engineers typically are confronted with four problems. (i) They have to deal with multiple artifacts concurrently, (ii) they have to work with artifacts that are too large for being displayed entirely on the screen(s) of their computers, (iii) the artifacts they are working with have multiple interconnections, and (iv) they have to work collaboratively with other requirements engineers as well as with stakeholders. The problem (ii) is particularly severe for large artifacts that do not have a linear structure. The problem (iii) is aggravated by the fact that proper information about artifact interrelationships is frequently not available. Hence, adequate tool support for coping with these problems is vital.

Our research aims at providing better tool support for requirements engineers when working collaboratively with many large and interconnected artifacts. As a first step, we wanted to gain an in-depth understanding of the state of practice in this area and draw conclusions about directions for improvement. Based on this goal, we defined four objectives for the research presented in this paper: (i) Examine the properties of requirements artifacts related to information presentation and how users work with different artifacts; (ii) Investigate the challenges related to information presentation that practitioners face when working with artifacts; (iii) Explore what methods practitioners use to overcome the identified challenges and how effective they are; and (iv) Study to which extent existing work addresses the identified challenges.

In order to learn about the actual challenges that practitioners are faced with when working with multiple artifacts and how they deal with these challenges (objectives 1-3), we designed and conducted an exploratory study where we interviewed 29 practitioners from eleven countries. After analyzing the interview data, we closed information gaps by conducting a follow-up survey.

For two reasons we decided to include all software development artifacts (including requirements, design artifacts, test cases, etc.), except source code in our study: first, the challenges of working with artifacts are not confined to RE. Second, recruiting participants that purely work with RE artifacts or can isolate their experience with RE artifacts from other artifacts was not possible. As a result, eight out of 29 interviewees are requirements engineers (business and software analysts), while the others are software architects, software developers, software testers, and project managers. We analyzed the interview data separately for every role and found that the results were not significantly different from those obtained for all interviewees (see Sect. 2.3.2). Hence we conclude that our results not only characterize how practitioners utilize tools to deal with various types of artifacts in general, but that these results are equally valid for how requirements engineers deal with requirements artifacts.

In addition to the challenges, we also have studied the workarounds that practitioners employ to deal with the challenges we have identified. We analyze how these challenges and workarounds may affect the effectiveness, efficiency, and satisfaction with which practitioners use tools to manipulate artifacts. Finally, we discuss

other studies and tools that provide support for working with artifacts especially when artifacts are many, large and interconnected (objective 4).

This paper is an extended version of a conference paper [GG16], where we presented our exploratory study with the first analysis of our findings. In this paper, we make four additional contributions:

1. We provide a deeper analysis of the interview data, resulting in several more findings;
2. We conducted a follow-up survey which provides answers to issues that were left open in our exploratory study;
3. We consolidate all our findings into a comprehensive view, analyzing how the found challenges impact the effectiveness, efficiency, and satisfaction with which the practitioners use tools to work with artifacts;
4. We discuss to which extent existing studies and tools address the identified challenges.

The remainder of this paper is organized as follows. We define some key terms in Section 2.2. In Section 2.3, we describe our research methodology. Our key findings are presented in Section 2.4. In Section 2.5, we consolidate all our findings into a comprehensive view. Existing work that addresses the challenges we found in our study is discussed in Section 2.6. Section 2.7 describes further related work. In Section 2.8, we explain the threats to the validity of our study and what we did to limit them. Section 2.9 concludes with a summary and outlook.

2.2 Definition of Terms

In this section, we define some key terms that we will use frequently in this paper. While these terms are being used broadly in the literature and daily life, we nevertheless provide definitions to avoid any misunderstanding caused by presumptions.

2.2.1 Practitioner

In this study, we consider *practitioners* to be professional persons who actively contribute to the development or evolution of a software-based system. A practitioner may possess any role such as requirements engineer, project manager, software architect, software developer, or software tester. In the study presented in this paper, we specifically aimed at practitioners who work with artifacts on a daily basis.

2.2.2 Artifact

In the context of this paper, an *artifact* is any kind of textual or graphical document with the exception of source code. Artifacts may be, for example, textual requirements documents, graphic models (including Unified Modeling Language (UML) diagrams), glossaries, charts, or sketches. Artifacts can have any size and granularity, ranging from comprehensive documents to user stories,

use cases, bug reports or diagrams. They may use various notations and can be interconnected to other artifacts. We excluded source code for two reasons: (i) we are primarily interested in artifacts relevant to requirements engineering, and (ii) the tools used for handling source code are different from those for handling other artifacts. When multiple elements such as textual requirements, user stories or diagrams are stored together in one document, we consider that document as a single artifact. Our rationale for doing so is the fact that the whole document needs to be opened, searched and navigated even when just a single element, such as an individual user story, needs to be retrieved or edited.

2.2.3 Tool

Any software product which is used for creating, editing, viewing, or managing artifacts is considered to be a *tool*. We are interested in tools that allow users to directly work with artifacts, i.e., viewing, navigating and editing them.

2.2.4 Screen

Regardless of the tools used for dealing with artifacts, information is presented to the tool users through display devices, for example, computer monitors, built-in screens of laptops and tablets, or electronic whiteboards. The size of these devices, i.e., the available space for displaying information in a readable size, imposes

limitations when working with artifacts. In this paper, we use the term *screen* to denote any device that a tool uses to present an artifact. Practitioners use screens of various sizes and may also use multiple screens of different sizes simultaneously. When analyzing our data, we use the aggregated screen size or the maximum screen size depending on our analysis purpose. The reason is explained whenever such a decision is taken. In this paper, an artifact is considered to *fit on a screen* when it is still readable after zooming out to become entirely visible. Fitting on a screen depends on the screen size, screen resolution and the eyesight of the user. In high-resolution screens, an artifact can be highly zoomed out while preserving adequate detail. However, the detail may be unreadable due to human eyesight limitations. Considering that high-resolution screens are being commonly used and, with the help of accessories if needed, there is not much difference in human eyesight, the most influential factor is the screen size.

2.2.5 Interaction when Working with Artifacts

Our work aims at better understanding how requirements or software engineers work with artifacts that are stored on their computers, i.e., how they access, search, navigate or edit artifacts with the help of *tools*, using common input/output devices such as screens, keyboards, and mouses. We study the *interaction* methods and mechanism that tools provide for enabling users to perform the access, navigation, manipulation and management actions on artifacts. Scrolling, zooming, resizing and switching windows are some

of the most frequently used interactions. We do not investigate the actual information being stored, accessed or manipulated in these artifacts.

2.3 Research Methodology

To investigate how practitioners work with artifacts and how they deal with the interaction challenges encountered in various situations, we conducted an exploratory study [Sea08] based on semi-structured interviews [KH10]. We were interested in situations such as when the artifacts do not fit on their screen, should be handled simultaneously or their relationship information is not kept well. Later we complemented the data from the interviews with a follow-up survey. In the interviews, we were able to collect quantitatively analyzable data and information about how practitioners actually deal with the challenges they encounter at the same time. The latter was made possible by asking open questions. Consequently, our data set is partially composed of qualitative data [Rob02]. The interview format also gave us the chance to explain the questions to the participants well enough to avoid misunderstandings and collect more accurate answers. The survey allowed us to reach a higher number of participants and fill the information needs that we encountered when analyzing the interview data.

2.3.1 Research Questions

From the goal and the objectives that we presented in the introduction, we derived four research questions (RQ1-RQ4). Although our research questions are framed in the context of requirements engineering artifacts, we actually studied the more general problem of the challenges encountered when working with software engineering artifacts (except source code). As explained above in the introduction, the rationale for this decision is that these interaction challenges are not confined to requirements engineering.

RQ1. *How do different interaction-related properties of artifacts, tools, and screens affect the effectiveness and efficiency of working with requirements artifacts?*

Artifacts have different properties. Different types of artifacts require different types of interaction. Working with artifacts in software tools is more challenging when the artifact has certain properties such as being larger than the screen or being connected to other artifacts so that the practitioners need to work on multiple artifacts at the same time.

Tools and screens can also have different properties that affect user interaction. The features of the tools, and the size of the screens are two of such properties. To have a deeper understanding of the interaction with artifacts and improve it, we decided to investigate the properties of artifacts, tools, and screens.

RQ2. *What interaction challenges do practitioners encounter when working with requirements artifacts?*

Secondly, we wanted to know the interaction challenges that practitioners are faced with when working with requirements artifacts of different properties, particularly with respect to the presentation of artifacts on the available screen(s). We encouraged the participants to tell us about all the challenges they encounter related to navigating, manipulating and managing artifacts (or any other action they perform on their artifacts).

RQ3. *What methods do practitioners use to handle the interaction challenges of working with requirements artifacts?*

Having identified the interaction challenges experienced by practitioners, we study how they try to overcome them, e.g., whether they use features of tools or improvise other workarounds.

RQ4. *Which existing solutions address the challenges of working with RE artifacts?*

Finally, after discovering the challenges of working with artifacts and the workarounds that practitioners employ to cope with them, we identify some of the solutions (e.g., techniques proposed in research or tools) that attempt to address the challenges we found in our answer to RQ2. The answer to this question provides us an impression about how much effort is already devoted to solve those challenges and encourages researchers to investigate the reasons why despite of the proposed solutions the challenges still exist.

2.3.2 Study Design

1) Initial Preparation: Our semi-structured interviews were based on an interview instrument[1], which was first elaborated as a list of questions linked to the RQs and the goals of the study. The interview instrument was designed following the guidelines stated by Oates [Oat05]. Then it was improved in two rounds of evaluation and feedback: first, it was evaluated by a group of RE experts to discover possible ambiguities and shortcomings. Second, we conducted two pilot interviews with a researcher from the University of Zurich and a practitioner.

The interview instrument comprises four sections: (i) characterization of the company and the interviewee, (ii) properties and types of artifacts used by the interviewees and the tools they use to handle them, (iii) challenges interviewees encounter when working with certain types of artifacts, and (iv) how they deal with these challenges.

2) Selection of Participants and Demographics: The 29 practitioners we interviewed can be categorized into five roles: we had eight *requirements engineers* (business and software analysts), five *software architects*, nine *software developers*, four *project managers*, and three *software testers* (Figure 2.1a).

Requirements artifacts are used almost in all phases of software development. To obtain a comprehensive view of tools and challenges related to artifact creation, modification, comprehension,

[1] http://www.ifi.uzh.ch/rerg/people/ghazi/InterviewInstrument2016.pdf

and management, the study cannot be constrained to a specific phase or role of software development. Otherwise, some of the mentioned aspects will be overlooked.

Many roles in software development work with multiple types of artifacts (e.g., requirements artifacts and design artifacts). Asking participants to consider only one type of artifact when answering the questions would result in inaccurate data since it is possible that the participants unintentionally consider wrong types for some parts of the questions. Assuming that documentation is done in a similar way in different phases of software development, we neither restricted our study to a particular development role, nor asked the participants to consider specific types of documents. In the analysis phase of our study, we searched for correlations between the roles and other parts of the data set (e.g., the size of artifacts and screens, various challenges), but did not find any. For example, Figure 2.9 shows how artifact size distribution is similar between different roles. This indicates that there are no significant differences among the different phases and roles of software development with respect to the questions we are exploring.

We defined two criteria to ensure recruiting suitable participants for our study. We looked for software development practitioners who (i) had at least one year of experience of being a member of a software engineering team, and (ii) used software and requirements artifacts (textual and graphical) on a daily basis during their work. The self-evaluation of the participants' experience in working with artifacts is shown in Figure 2.1b. The largest group of participants (38%) reported between four and seven years of experience and 14%

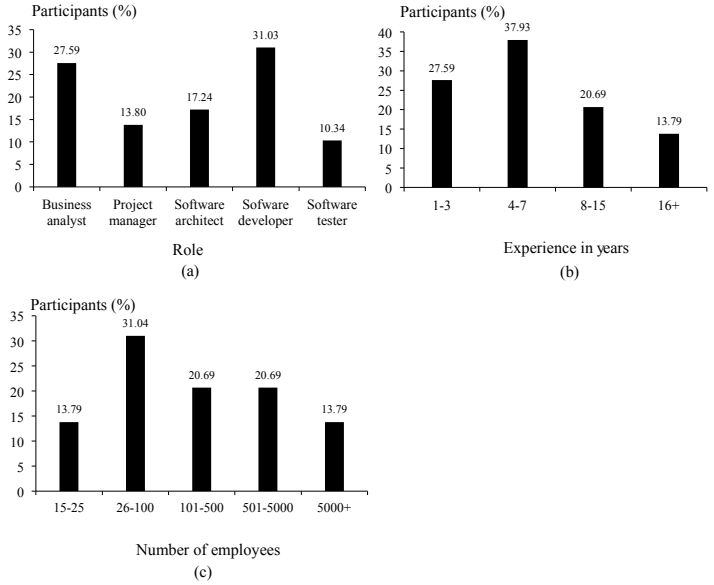


Figure 2.1: Distribution of participants with regard to (a) distribution of the roles (b) years of experience and (c) size of the company measured in number of employees

had more than 16 years of experience in working with requirements and software artifacts.

With respect to company size, our study covers a wide range, from under 25 to more than 5000 employees. The largest group of participants works in companies having between 26 and 100 employees (31%) and 14% having more than 5000 employees. Figure 2.1c presents this distribution.

When recruiting participants, we combined two types of sampling: snowball and convenience sampling [KP02]. For the snowball

sampling, we started from our immediate acquaintances who were active in any phase of software development such as requirements engineering, software design, software development and testing. We sent an e-mail containing a short description of the research and the selection criteria mentioned above, making it clear that participation is voluntary. We asked them to send us a short description of their duties and whether they work with both textual and graphical artifacts or not. After each interview, we asked them to introduce other practitioners who fit the criteria. The majority of the participants believed that another interview with a person from their company would result in similar data. Although the redundant data could help us in validating the data set gathered, we decided to invest our resources on increasing the variety of the participants, and recruit the next group of participants from the acquaintances of the first group working in other companies. The majority of our participants (86%) were recruited by snowball sampling. For the convenience sampling, we used a social network of professionals (LinkedIn). We sent LinkedIn messages (InMails) to a randomly selected set of practitioners and asked them whether they comply with our selection criteria.

Eventually, we interviewed 29 practitioners from 29 different companies, located in eleven countries from seven geographical regions, as depicted in Figure 2.2.

3) Data Collection and Analysis: The interviews were conducted between November and December 2015. Their duration varied between 40 and 87 minutes, with an average of 56 minutes. We conducted the interviews over Skype or Google Hangouts, except

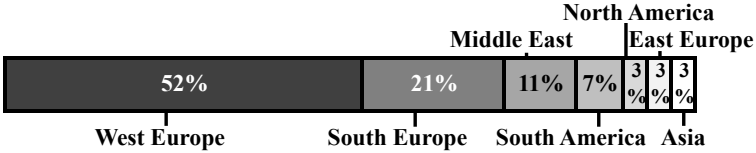


Figure 2.2: Geographical distribution of participating companies

for three, which were conducted face-to-face. All interviews were recorded and transcribed completely.

We used Microsoft Excel and MATLAB in the data analysis phase. We started to analyze quantitative data by first extracting the simple quantitative variables such as a number of screens and second computing the complex variables such as average artifact size for each participant. To analyze the qualitative data we employed a qualitative analysis approach [Cre13] comprised of six steps: (i) preparing the raw data in spreadsheets for analysis, (ii) extracting different items from each answer of each participant, e.g., the mentioned challenges, (iii) providing a description and a code for each item, (iv) identifying the similar items by their descriptions and grouping them, (v) restructuring data based on the identified groups and counting the frequencies, and (vi) finalizing our findings.

4) Follow-up Survey: Ten months after the initial interviews and its data analysis resulting in several findings, we carried out a follow-up survey to gather the information that we needed for further potential findings. Particularly, we wanted to (i) have more detailed information about storing artifact interrelationships, and

(ii) know which productivity factors to consider when analyzing the challenges of handling artifacts. The survey came in two formats: as an editable pdf file[2] and as a Google Form. The participants could use one of these formats based on their preferences. The responses were gathered within ten days. To measure the attitude of the participants, we used a five-point Likert scale [Lik32].

We sent the survey to 59 people and successfully received 42 responses, yielding a response rate of 71%. We made sure that all participants answered willingly. 20 participants had also participated in the initial interviews. The remaining participants belong to two groups: (i) fourteen people whom we had identified as potential interviewees, but eventually did not interview due to their or our availability constraints, and (ii) eight persons working in academia as requirements engineering researchers whom we knew and could approach directly. Obviously, the participants in the former group meet the selection criteria we had defined for interview participants. The members of the latter group all had the knowledge required for answering the survey questions. In total, as Figure 2.3a shows, we received responses from academic experts (19%), software developers (28%), software architects (10%), project managers (12%), software testers (7%), and requirements engineers(24%). Among the academic experts, we had one professor, two post-doctoral researchers, three Ph.D. students and two graduate students (Figure 2.3b).

All academic participants are active in the field of requirements engineering. The majority of the industry practitioners have four

[2]http://www.ifi.uzh.ch/rerg/people/ghazi/Follow-upSurvey2016.pdf

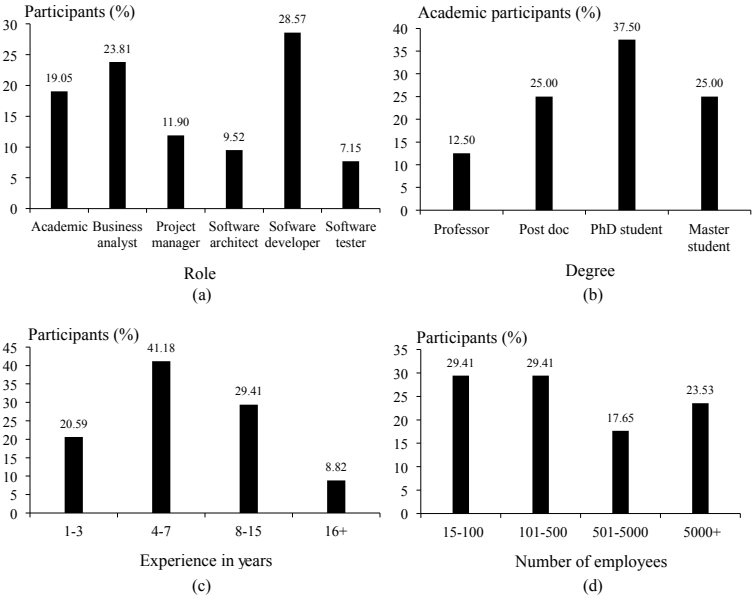


Figure 2.3: Demographic information about the survey participants: (a) distribution of the roles, (b) the level of experience for academic participants measured by their academic degree, (c) years of experience in working with artifacts, (d) size of company for industry practitioners measured in number of employees

to seven years of experience (41%) and 9% have more than sixteen years of experience in working with artifacts (Figure 2.3c). The largest group of participants are working in companies having between 15 and 100 (29.4%) employees or between 101 and 500 employees (29.4%). The remaining participants work in large companies; see Figure 2.3d.

2.4 Key Findings

In this section, we present and discuss the findings from our interviews and the follow-up survey, thus answering our research questions RQ1-RQ3. We coded our findings with respect to their corresponding research questions: FA-FC respectively. For example, the first finding regarding RQ1 is coded FA1. For every finding, we provide the evidence from the interviews and the survey, and a short discussion of the importance of that finding. We will address RQ4 in Section 2.6.

As our interview study is based on a relatively small, non-representative sample, the quantitative figures we report in our findings must be treated with care: from a statistical viewpoint, these quantities are not generalizable. While the quantitative data we report are factual for our sample, a study based on a representative sample might yield different results. Hence, with respect to generalizability, our quantitative findings should be considered as hypotheses, rather than generally valid facts. Nevertheless, we believe that a quantitative evaluation of our study data adds value for the readers of this paper.

2.4.1 *Interaction-related Properties of Artifacts (RQ1)*

In a first step, we wanted to learn about the interaction-related properties of artifacts, screens, and tools. We investigated artifact

properties such as the size of artifacts, the maximum number of artifacts used simultaneously and the notation used in the artifacts. We also asked about the size and the number of screens and the software tools used by the participants.

Finding FA1. Only about one-third of the graphical artifacts used by the interviewed practitioners fit on their screens.

Evidence for FA1. In the interviews, we explicitly asked the participants about the percentage of their software engineering graphical artifacts that fit on their screen according to the definition of fitting provided in Section 2.2.4. We explained to the participants to consider the situation where their artifacts are zoomed out as much as possible just before the details get too small to be recognized by their eyes. We assumed that the resolution of their screens did not limit the zooming. Figure 2.4 visualizes this information. In this question, we made a distinction between textual and graphical artifacts because textual artifacts have their own way of navigation, search, and management. The bounding box represents the entire set of the participant's artifacts. This space is partitioned into 29 vertical bars. Each bar represents a participant and is filled according to the participants' answer. We sorted the participants based on the percentages in order to have the filled areas on one side and not-filled areas on the other side. The total not-filled area in the resulting chart (Figure 2.4) represents 65% of the artifacts that do not fit on the respective participants' screens. This percentage can be ascertained by the ratio of the light gray area to the whole area of the box.

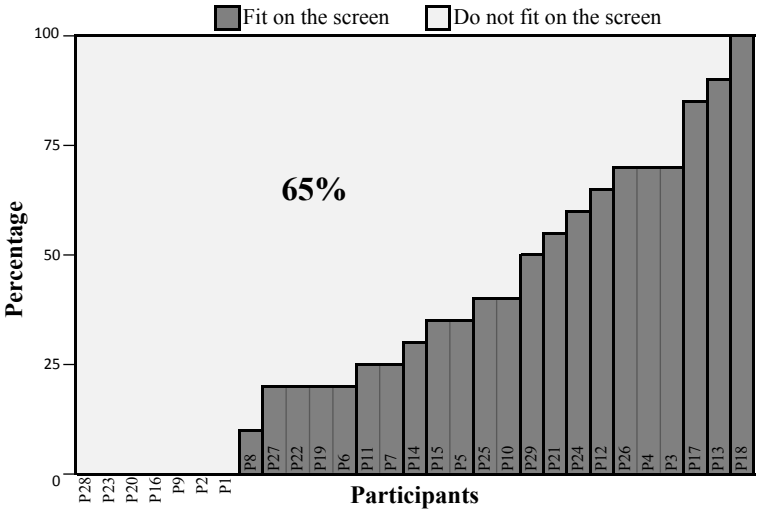


Figure 2.4: Percentage of software engineering graphical artifacts that fit on the participants' screens

Discussion. The size of artifacts affects many aspects of working with artifacts. Even when only a part of an artifact is needed for a task, a practitioner needs to search for the part of interest, navigate through the artifact to reach that part and adjust the zoom level according to the given task. Also, when practitioners modify some part of an artifact, they nevertheless need to view and consider the whole artifact in order to preserve the consistency of the artifact.

One can argue that the participants' screens have different sizes and the percentages found depend on the actual screen sizes. Although this argument is true, the chart still shows the percentage of the artifacts that are being used on screens that do not permit to view

these artifacts entirely. In the next finding we have eliminated this dependency.

Finding FA2. About forty percent of the graphical artifacts do not fit on the largest screen reported in this study.

Evidence for FA2. We wanted to investigate the artifact size in a way that does not depend on the participant's screen size. The difficulty was that no common measure for artifact size exists that everybody understands and that allows comparisons. To overcome this problem we made two decisions. (i) As the measure for artifact size, we decided to use the smallest screen size on which the artifact fit according to our definition of fitting from Section 2.2.4. (ii) Since it was nearly impossible to ask participants to describe all of their artifacts with this measure, we asked a simpler question that led us to find the distribution of their artifacts' sizes. In particular, we asked the participants to estimate, in percent, the amount of their artifacts that fit on screens of four different sizes: 11-inch, 15-inch, 22-inch, and 28-inch. This question not only is simpler but also provides more information.

We decided to use the estimations for 15-inch and 22-inch screens since, according to the frequency of screen sizes illustrated in Figure 2.5, the 22-inch screen is the most used, followed by the 15-inch screen. We also used the percentages provided for 28-inch screens to include the percentage of the artifacts that fit on the largest monitor (according to this research). However, we dropped the data for 11-inch screens as only one participant used a screen of this size. The total number of screens is higher than

the number of participants because many of them had more than one screen.

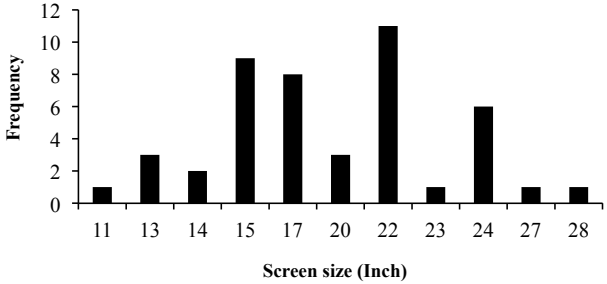


Figure 2.5: Distribution of screen sizes of the participants

We visualized the data gathered from this question in the same format as Figure 2.4 which resulted in finding the percentages of the artifacts that do not fit on the above-mentioned screen sizes. According to our finding, 24% of the participants' artifacts fit on a 15-inch screen, 42% of the participants' artifacts fit on a 22-inch screen, and 58% of the participants' artifacts fit on a 28-inch screen. Figure 2.6 shows the percentages of graphical artifacts fitting on a 28-inch screen. The bounding box represents the entire set of the participants' artifacts. This space is partitioned into 29 vertical bars. Each bar represents a participant and is filled according to the participant's percentage of artifacts that fit on a 28-inch screen. The empty space (light gray) which constitutes 41% of the whole box shows the overall percentage of the artifacts that do not fit on a 28-inch screen.

Discussion. In Figure 2.6, the part with the lightest gray color has taken more than 41% of the area and represents the artifacts that

do not fit even on the largest screen reported in this study. This result shows that supplying larger screens will alleviate the problem by fitting more artifacts but not solve the problem completely: alternative solutions are needed.

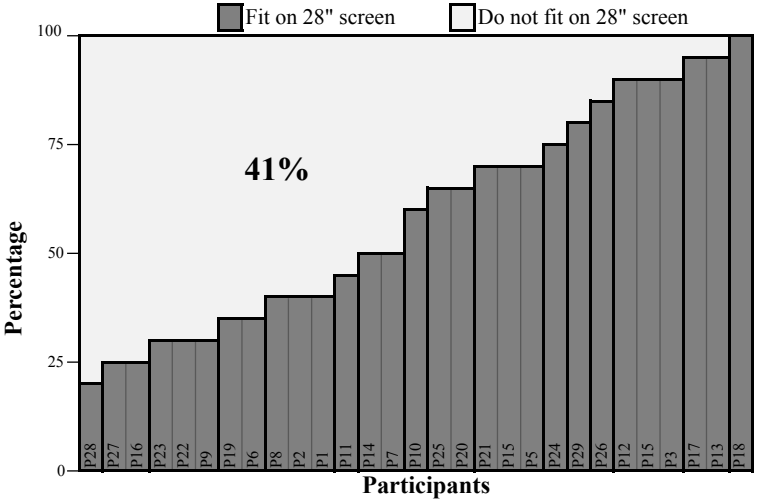


Figure 2.6: Percentages of graphical artifacts that fit on a 28-inch screen

Finding FA3. More than half of the interviewed practitioners use four or more artifacts at the same time.

Evidence for FA3. We asked the participants about the maximum number of artifacts they use simultaneously. We received different answers ranging from one to twenty. The box plot in Figure 2.7 shows the distribution of the answers. Half of the participants use between three and six artifacts at the same time; the median is four. Two practitioners with twelve and twenty artifacts

used at the same time are outliers, therefore we can say that the number of artifacts being used simultaneously ranges between one and ten. The types of the simultaneously used artifacts can be similar or different. For example, P15 was a lead business analyst working with ten artifacts at the same time of the following types: functional and nonfunctional requirements specification documents, scope specification documents and diagrams such as use case, sequence, flowchart, BPMN, and mindmap.

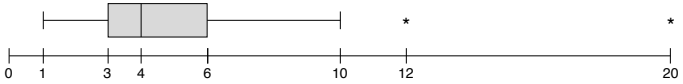


Figure 2.7: Number of artifacts used at the same time

Discussion. This finding emphasizes the importance of screen space and how it is used to present artifacts. From the fact that a significant portion of artifacts are larger than the available screens (from FA1 and FA2) and more than half of the practitioners use four or more artifacts at the same time, we conclude that practitioners either need larger screens (which is limited by cost and technology) or the existing screen space must be used in a smarter way when presenting artifacts to their users. The challenges of working with multiple artifacts at the same time are discussed in FB2.

Finding FA4. More than two-thirds of the interviewed participants use customized notations for their artifacts.

Evidence for FA4. We asked our participants which notations they use for their artifacts. Except for structured text which is used by everyone, the majority of practitioners mentioned that

they use the UML. Further, more than half of the participants also use customized notations. Three participants (10%) reported that they use the Business Process Model and Notation (BPMN) as well. Figure 2.8 displays the distribution of notations used for graphical artifacts by practitioners. The sum of the percentages is greater than 100% since many practitioners use more than one notation at the same time.

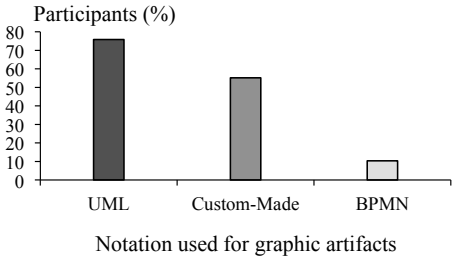


Figure 2.8: The distribution of graphic notations used by participants

Discussion. This finding shows that although UML is the dominating notation in the software development industry, companies also customize some notations based on their needs. Therefore, any assumption about the properties of artifacts (e.g., size, number, and interconnectivity) based on UML notation or other well-known notations may result in developing non-generalizable interaction methods. Such a tool may limit its user in performing certain tasks or may force its user to perform unnecessary tasks for the custom notation chosen. This is one of the reasons why practitioners use other means (for example, pen and paper or whiteboards) to create graphic artifacts (this will be discussed in more detail in FC5).

Finding FA5. On average, practitioners use more than five applications concurrently to create and manage their artifacts.

Evidence for FA5. We asked our participants about the artifact-related applications they use and compiled a list of the applications mentioned. This list includes well-known applications as well as less popular ones. The point that caught our attention was that our participants used many applications concurrently. For instance, one of the participants mentioned ten applications that he used in each project. The tools can be used simultaneously or at different points of time. Practitioners use multiple tools at the same time when multiple artifacts are needed simultaneously and when they need multiple artifacts that require different tools to be displayed and manipulated.

Discussion. According to our interviews, using multiple software tools concurrently is common in the software development industry and practitioners are used to it. In this regard, the following points may cause challenges in working with artifacts or exacerbate other challenges: (i) Practitioners must continuously switch between the applications that have different ways of interaction. The challenges of switching between windows are discussed in FB2. (ii) A new member of the team has to learn how to work with all these different applications, which causes a steep learning curve. (iii) Practitioners have to keep track of the relationships between artifacts manually, which is discussed in FC7.

Finding FA6. Although almost all participants work on their artifacts collaboratively, less than one-third of the collaboration is done with dedicated software development tools.

Evidence for FA6. Except for one of the participants, all interviewees mentioned that they collaboratively work on artifacts. According to our study, many different tools are used for allowing multiple practitioners to work on one artifact at the same time. We categorized these tools into four categories: file sharing (e.g., shared folder, Google Drive), view-only sharing (e.g., shared monitor, video conference tools), general collaboration tools (e.g., SharePoint, wiki pages) and software development-specific tools. Table 2.1 shows the percentages for each category and some examples. 28% of the mentioned tools are in the software development specific category and the majority of the collaboration (45%) is done by view-only methods. 17% of the mentioned collaboration tools belong to the general collaboration tools category, e.g., Google Docs and wiki pages. The remaining 10% of the collaboration is done by simple file sharing.

Discussion. The information from the interviews clearly shows that practitioners work on artifacts collaboratively. The collaboration may be a discussion about the artifact over a video conference or creating the artifact on a whiteboard. Many software development tools support collaboration directly or via plugins. However, there are many practitioners who still use methods such as sharing a file, which has major problems such as the danger of overwriting each other's work. Another group of practitioners uses general-purpose collaboration tools such as wiki pages or SharePoint, which reduce the problems of file sharing but still need extra work to import the artifacts and keep them up-to-date. This is because the editing capabilities of this type of tools are inferior to tools

specifically built for the purpose of editing software development artifacts.

Table 2.1: The percentages of different categories of collaboration tools and examples for each category

Category	Percentage	Examples
View only sharing	45%	Shared monitor, video conference tools, whiteboards
Software development specific tool	28%	JIRA, Confluence, Enterprise Architect
General collaboration tool	17%	SharePoint, wiki pages, Google Docs
File sharing	10%	Shared folder, Google Drive

2.4.2 *Challenges of Working with Artifacts (RQ2)*

After asking the participants about the properties of their artifacts, screens, and tools, we asked about the challenges they experienced when working with artifacts larger than their screens, the challenges they face when working with multiple artifacts at the same time, and the challenges of dealing with the interrelationships between artifacts.

Finding FB1. "Relying on memory", "Searching for information", and "Maintaining the overview" are the most important challenges in handling large artifacts.

Evidence for FB1. After gaining a perspective of the participants' artifact size and screen size, we asked them about the challenges of working with artifacts that are larger than the available screen. After gathering all challenges, we first created a

comprehensive list of them. To guarantee atomic and concrete challenges, we removed general ones (e.g., "*Working with large artifacts is not efficient*") and dependent ones (e.g., "*This type of artifact takes so much time*"). Afterwards, we grouped similar challenges that were expressed in different words. For example, below we give some quotes about how participants rely on their memory. "*[P23:] It increases your cognitive overhead because you do not remember where things are*", "*[P10:] You have to imagine what is located in the part of the picture you cannot see*" and "*[P14:] Because I forget things easily I have to take notes in another place*". Table 2.2 presents the list of challenges and their frequency (number of participants mentioning that challenge). The calculation of the priorities is explained below.

Table 2.2: Challenges of working with artifacts that are larger than the available screen. Prioritization is explained in the text.

ID	Challenge	Priority	#Participants
C1.1	Relying on memory	1	18
C1.2	Searching for information	2	2
C1.3	Maintaining the overview	3	20
C1.4	Too much scrolling & zooming	4	29
C1.5	Knowing the current location	5	5
C1.6	Following the links	6	7

The frequency of mentioning a challenge alone is inadequate to show the importance of the challenge, as different participants may be affected by the challenges to different extents. Therefore, we decided to rank the participants based on how heavily they are affected by the challenges of working with artifacts that are larger than their screen(s) and to use this ranking for prioritizing

the challenges listed in Table 2.2. For this purpose, we assumed that people who have larger artifacts and smaller screens are more challenged than others since they have larger parts of their artifacts outside of their screens. For example, in such a situation, more scrolling and zooming are needed, searching is more difficult and more information is needed to be kept in the mind.

As a first step, we computed the average artifact size of each participant using the Cumulative Distribution Function (CDF). We gathered three points for the CDF of the artifact size for each participant: the participant's estimation of which percentages of artifacts fit on 15-inch, 22-inch, and 28-inch screens, respectively (according to FA2).

CDF is calculated by the following formula [MM07]:

$$CDF(x) = \frac{1}{2}\left[1 + erf\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right] \tag{2.1}$$

x represents the screen size and $CDF(x)$ is the percentage of artifacts that fit on a screen of that size. These values are known from the interviews. μ and σ are mean and variance respectively. erf is called error function and is already known [MM07]. So we rewrite the formula as:

$$\sqrt{2}\, erf^{-1}\left(2\, CDF\left(x\right) - 1\right)\sigma + \mu = x \tag{2.2}$$

This is a linear equation with regard to the parameters σ and μ. Therefore, by plugging the mentioned three points, we can

calculate σ and μ. In the rest of this paper, the calculated μ is called S_μ. Since it is the mean of the artifact sizes for each participant, it indicates "the size of the smallest screen that can accommodate half of the artifacts in a readable size". Figure 2.9 shows the overall and role-wise distribution of S_μ.

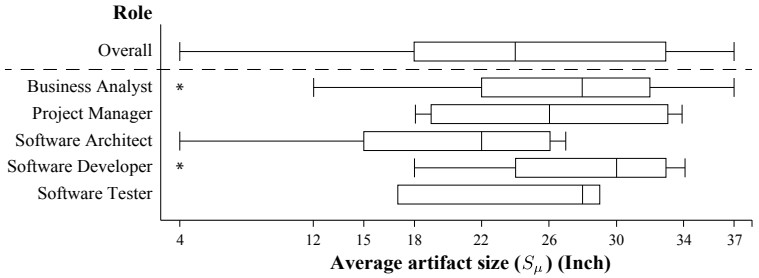


Figure 2.9: Overall and role-wise distribution of the average artifact size (S_μ)

Having calculated S_μ for all participants, we now can rank them with respect to the degree that they are affected by the challenges of working with artifacts that are larger than the available screen(s). We do this by assigning a point to each participant in a coordinate system with screen size as the x-axis and S_μ as the y-axis, and calculating the Euclidean distance of each point from a reference point which represents a hypothetical person who is more under influence of these challenges than all others in our dataset. This hypothetical person has a screen size of 12 (less than everyone else) and a S_μ value of 40 (more than everyone else). The result is depicted in Figure 2.10. The labels show the number and rank of the participants. The closer a participant is to the reference point on the top left edge, the more she or he is affected by

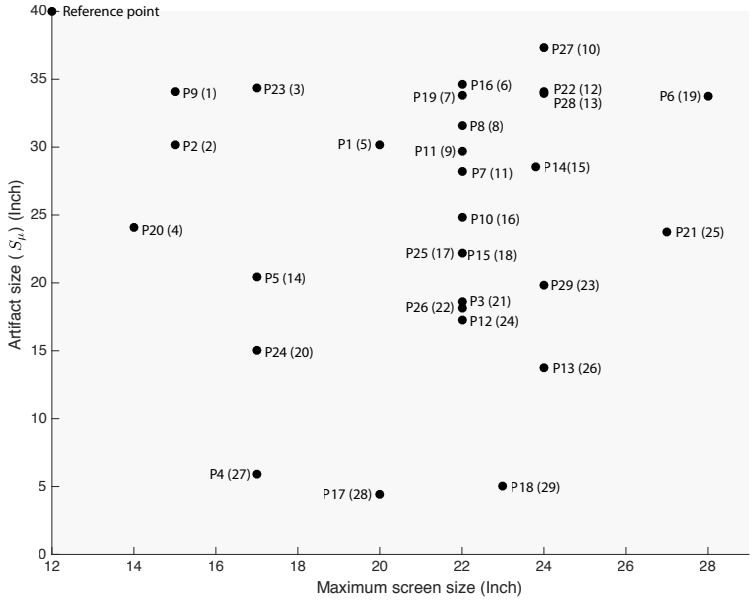


Figure 2.10: Ranking of the participants with respect to artifact and screen size used. The ranks of the participants are determined by measuring their Euclidean distance from the reference point.

the challenges of working with artifacts that are larger than the available screen(s). The rank of participants who have more than one screen was calculated based on their largest screen, assuming that they work with large artifacts on their largest screen.

We sorted the participants based on the calculated distances and ranked them so that the participant with the lowest distance has the highest rank of 1 and the participant with the highest distance has the lowest rank of 29. Then we propagated the ranks of the

participants to the challenges they mentioned. For example, if X is a challenge mentioned by three participants with ranks (1, 8, 18), we ranked X with the average of the participant ranks, which is nine. Finally, we prioritized the challenges according to their ranks as shown in Table 2.2.

Discussion. The most frequently mentioned challenge is the problem of too much scrolling and zooming (C1.4), which is an obvious impact of working with an artifact that is larger than the screen. However, after prioritization, the top challenge is the participants' need to rely on their memory (C1.1), particularly to avoid excessive scrolling and zooming (C1.4). "*[P27:] I use my memory if I can avoid zooming and scrolling around*". Maintaining the overview over the artifact (C1-.3) is the third top challenge with respect to the number of participants who mentioned this challenge. Interestingly, searching for information (C1.2) is the second ranked challenge, although it was mentioned by two participants only. "*[P3:] When you want to find information in an artifact and the artifact is a big one, it is very hard. Searching information in larger artifacts is harder*". This illustrates to what extent searching for information can be cumbersome when having large artifacts and a small screen.

Finding FB2. "Switching between windows" and "Working in too small windows" are the most important challenges when working with multiple artifacts.

Evidence for FB2. The challenges of working with multiple artifacts are identified and prioritized similarly to FB1 In this case,

we identified three related parameters for ranking the participants: screen size, S_μ (smallest monitor size that accommodates half of the artifacts) and the number of artifacts used at the same time. We assigned a point in the three-dimensional space to each participant. We assumed that a person with large artifacts, a small screen and a high number of concurrently used artifacts is stronger impacted by the challenges of working with multiple artifacts than others. Therefore, we calculated the Euclidean distance between each participant point and the reference point showing a hypothetical person having an extremely small screen, a S_μ value of 40 (higher than everyone else), and using 20 artifacts at the same time (the highest in our dataset). When analyzing, we aggregated the screen sizes of participants who use multiple screens, since they can open different artifacts on different screens at the same time and use all of the available screen space. The result is depicted in Figure 2.11. The labels show the number and rank of the participants and the reference point denotes a maximally challenged hypothetical person. Note that the actual geometric distances of the points are different from what they seem to be in this figure since, for better readability of this illustration, we are using different scales on the three axes.
We used the calculated distances to rank the participants. Then we propagated the ranks of the participants to the challenges they mentioned and sorted the challenges based on the average of their ranks. Table 2.3 shows the result.

Discussion. Practitioners mostly use multiple screens and multiple windows to work with multiple artifacts at the same time. According to the participants, the most cumbersome task is switch-

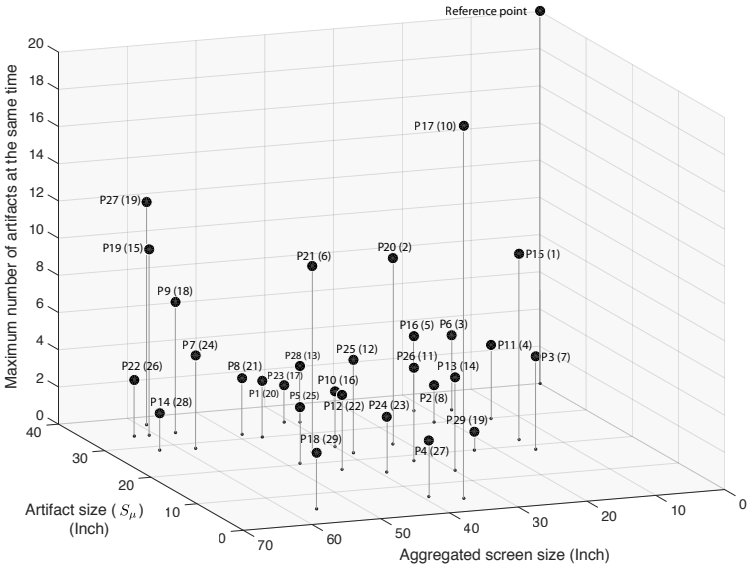


Figure 2.11: Ranking of the participants with respect to artifact size, screen size and number of artifacts used. The ranks of the participants are determined by measuring their Euclidean distance from the reference point in the three-dimensional space.

ing between windows (C2.1), and it gets worse when the tools only support multiple tabs (instead of multiple windows). To make navigation between artifacts easier, some of the practitioners arrange windows side by side. This results in easier switching but raises two other challenges: (i) Each artifact has less dedicated space, therefore the user has to work in a smaller window (C2.2). "*[P16:] Most of the time working with windows side by side is really difficult because there is less space to work*". (ii) The windows need

Table 2.3: Challenges of working with multiple artifacts at the same time. Prioritization is explained in the text.

ID	Challenge	Priority	#Participants
C2.1	Switching between windows	1	23
C2.2	Working in too small windows	2	12
C2.3	Changing focus	3	8
C2.4	Knowing the relations between artifacts	4	6
C2.5	Finding the right window	5	16
C2.6	Arranging windows	7	4
C2.7	Memorizing	8	10
C2.8	Finding the current focus position	9	2

arrangement (C2.6). "*[P22:] I arrange the windows regularly and resize them. When you expand or close a window you have to move the other windows*". When moving back and forth between multiple artifacts, practitioners may forget the exact location where they left an artifact and need some time to find the right location when they return.

Therefore changing focus causes interruption in their work (C2.3). Finding the window that they need in a particular moment is a challenge (C2.5), since windows tend to hide under each other and using keyboard shortcuts to move through windows consecutively takes time and is error-prone. Recent operating systems (e.g., OS X) provide an overview of the open windows, but this becomes increasingly useless when a large number of windows is open. Moreover, bringing up the overview still takes time and does not work with multi-tab systems. When using multiple windows, there is no information about the relationship between the artifacts inside the open windows (C2.4). Since the space for each artifact is smaller, more information is located outside of the screen. Therefore, the users have to keep more information in their mind (C2.7). The

location of the cursor is easily mistaken when multiple windows are open (C2.8). Using more than one screen alleviate some of these challenges to some extent, e.g., switching between windows and memorizing. However, this solution does not resolve those challenges completely and leaves the other ones unaffected.

It also may give rise to new challenges such as switching focus between screens and remembering in which screen which artifact resides.

Finding FB3. Storing insufficient artifact relationship information provokes creating larger artifacts and makes searching and understanding artifacts more demanding.

Evidence for FB3. We found that practitioners do not store the relationships between different artifacts in a systematic way, because (i) software tools do not keep relationship information between artifacts (FC7) or, (ii) the artifacts are managed in different tools (FA5), which makes interconnections very difficult. At the same time, we found that most of the participants work on multiple artifacts simultaneously (FA3), which shows that artifacts are indeed interrelated. This motivated us to know what benefits would be gained if a comprehensive interrelationship view of all artifacts were possible. Therefore, we looked for the challenges that practitioners faced due to the lack of relationship information. In particular, we looked for sentences such as: " *[P16:] If I know the relationship between the artifacts of the project, I can imagine a large picture of the artifacts in my mind and I can organize my tasks*" or "*[P11:] When I want to understand an*

artifact, I spend half of my time on finding the relating artifacts". We then consolidated the challenges found into six groups (see Table 2.4).

Table 2.4: Challenges of not storing interrelationship information properly. Prioritization is explained in the text.

ID	Challenge	Priority	#Participants
C3.1	Forced to create large artifacts	1	2
C3.2	Search for information	2	5
C3.3	Inconsistent change	3	10
C3.4	Difficulty to understand	4	14
C3.5	Loss of the big picture	5	16
C3.6	Reliance on memory	6	2

The most frequently mentioned challenge (52%) is that practitioners lose the big picture of the artifacts (C3.5). They do not know the order of artifacts and which artifacts are complementary to each other. Consequently, they are less organized in handling the artifacts. The second most mentioned challenge (48%) is that understanding the artifacts gets more difficult when there is no information available about the related artifacts (C3.4). This challenge is particularly obvious when a new member joins a team. The new member will be disoriented when receiving a stack of artifacts without any relationships. Inconsistency can be caused by unawareness of the artifacts' relationships, as mentioned by 34% of the participants (C3.5). A practitioner can easily miss one of the related artifacts when changing an artifact. 17% of the participants stated that searching for information needs more time when the relationships are not known (C3.2). Two participants (7%) pointed out that they have to use more memory when they do not have a good method to store relationship information (C3.6). Finally, two participants specifically mentioned that not having a

decent way of storing relationship data forces them to create very large artifacts to avoid more relationships (C3.1). One of them stated "*[P9:] Generally I do not like to create several artifacts because when I want to change something I should be careful to change all of them. I prefer to have them in one place*".

Discussion. As with the previous challenges (FB1 and FB2), we prioritized the challenges we found. For this purpose, we assume that a practitioner with more interconnected artifacts and less effective methods to store relationship information is more affected. From the interviews, we knew the methods of storing interrelationships between artifacts that each participant used. In order to compare the effectiveness of those methods, we included a question in the follow-up survey and asked the survey participants how much relationship information each method captures. The participants had to rate each method on a five-point Likert scale. In addition, they had space to write any method that did not exist in our list. The responses to the open part of the question were methods that basically belonged to one of the items of our list, e.g., "traceability matrix" which is an extra artifact and "tagging" which is a tool feature. Table 2.5 shows the effectiveness of storing interrelationship information from the viewpoint of the survey participants after quantifying.

In Figure 2.12, we defined a two-dimensional space based on two parameters: (i) the effectiveness of storing relationship information and (ii) the number of artifacts used at the same time (discussed in FA3). To calculate a value that shows the effectiveness of the methods that each participant used for storing relationship information,

Table 2.5: Effectiveness of storing interrelationship information

Storing interrelationship method	Mean of effectiveness
Software tool	4.0645
An extra artifact	3.5806
References to other artifacts	2.9354
Folder structure	2.5161
File name convention	2.3225
Memorizing	1.9032

we first extracted the mean value for effectiveness of each method from the survey results. Then, using these extracted values, we calculated mean values for the participants based on the methods they mentioned in the interview. In Figure 2.12, each point represents a participant. The reference point represents a hypothetical person who has the highest number of connected artifacts (20) and the least effectiveness of storing relationship information encountered in this study (1.9). The Euclidean distance between each participant and the reference point shows how much that participant is affected by the challenges caused by poorly stored artifact relationship information. Note that the actual distances of the points are different from what they seem to be in this figure since, for better readability of the diagram, we are using different scales on the two axes. We ranked the participants based on their distances and used the ranking to prioritize the challenges. The labels show the number and rank of the participants. At the top of the prioritized list of challenges, we have a challenge which is mentioned only by two participants. This shows that the mentioned challenge is not of less importance than the others. In fact, this type of challenges that are important and mentioned by few participants are more difficult to track down. In our case, having no good means of storing relationship information unconsciously

forces the practitioners toward creating larger artifacts (C3.1). Large artifacts are difficult to understand and give rise to other challenges we discussed in this paper (FB1).

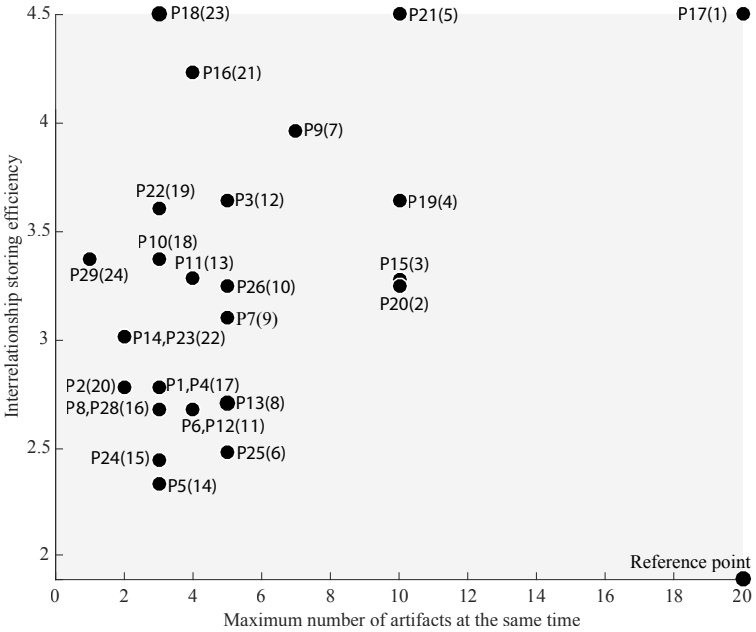


Figure 2.12: Ranking of the participants with respect to artifacts used at the same time and the effectiveness of the method they use. The ranks of the participants are determined by measuring their Euclidean distance from the reference point.

2.4.3 *Dealing with the Challenges (RQ3)*

We asked the participants how they deal with the challenges of handling artifacts. In particular, we were interested in knowing what methods they use to overcome the challenges mentioned above, what features their tools provide, and what they do when the tools do not support them sufficiently.

Finding FC1. Practitioners use their memory extensively.

Evidence for FC1. One of the main challenges in working with large artifacts, according to FB1 and FB2, is "relying on memory". To know how many of the participants use their memory intentionally, we asked them directly whether they memorize any part of an artifact to use it elsewhere or not. 69% of the participants answered affirmatively. We asked the participants who responded "no" how they handle the situations where they need a piece of information from another artifact or another part of the current artifact which is not on the screen at the moment. We found that 21% of the participants use copy and paste functions instead of keeping information in memory. They also mentioned that this method is not applicable to graphical information easily and they sometimes have to take screenshots. "*[P19:] I don't memorize. Instead, I use copy-paste. If it is a diagram, I would make a screenshot of it*". Finally, only 10% of the participants answered "no" decisively.

We asked the participants if better visual memory positively influences their performance. 82% of the participants admitted

that better visual memory affects their performance positively in working with graphical artifacts.

Discussion. The answers to these two questions prove that practitioners rely on their memory extensively when working with artifacts and show that they compensate their inadequate memory power by using copy-pasting and taking screenshots, which is error-prone and time-consuming in turn. Larger screens and additional screens are helpful to reduce the amount of information that practitioners keep in their memory. However, the screen size is limited and in many situations one artifact is not possible to be opened multiple times. In addition, working with an artifact spanned over multiple screens is challenging in turn.

Finding FC2. Traditional zooming and scrolling are the dominating techniques for handling large artifacts.

Evidence for FC2. As stated in FA1 and FA2, practitioners often work with artifacts that are larger than their screens. Many artifacts even do not fit on the largest screens reported in this study (FA2), i.e., they cannot view the entire artifact on the screen after zooming out in a readable size. However, when we asked them how they handle such artifacts and what features tools provide for this purpose, we found that they mostly use simple traditional methods such as scrolling and zooming.

Cockburn [CKB09] categorized visualization techniques that help handling larger-than-screen artifacts into four classes: zooming, overview + detail, focus + context and cue-based techniques. We

asked our participants how they handle large artifacts to know the techniques implemented in commercial tools. We found that traditional zooming and scrolling are the most basic techniques used for this purpose. In addition to zooming and scrolling, only three participants use tools that provide an overview + detail feature. The applications they use show an overview of the artifact in a small window and they can navigate inside the artifact by using this small overview. None of the interviewees have any focus + context or cue-based techniques available. Obviously, the features that exist, but are not known by the participants are not counted in this report. Three interviewees explained that they avoid having large artifacts by defining different layers of abstraction. The result is visualized in Figure 2.13.

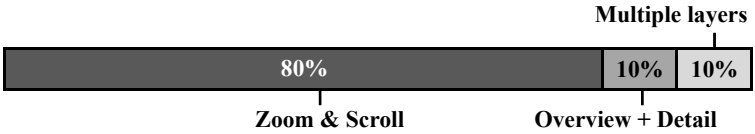


Figure 2.13: The percentages of the methods that participants used to handle artifacts that are larger than the screen

Discussion. By maintaining multiple layers of abstraction, some of the participants could manage to have smaller artifacts at the cost of increasing the number of artifacts and having redundant data in multiple artifacts. Managing a larger number of artifacts with redundant data needs additional effort. Working with multiple artifacts gives rise to other issues that we discussed in FB2. In this regard, participant 18 describes his needs as "*In my tool, different diagrams which show different layers of abstraction cannot be interconnected. What I really like is to start on a high level*

and go to a really detailed level, and get to the other diagrams that show the layers beneath. I do not know any software that has this kind of zooming".

Finding FC3. Almost all practitioners need to have an overview of the artifacts.

Evidence for FC3. The participants told us that they need to have a rough image of the entire artifact when working on it. This image makes them aware of the size, complexity, different sections, position of the important elements, and the elements' types of an artifact. 65% of the participants keep an overview of the artifact in their mind and 32% of the participants employ other techniques to maintain an overview of the artifact (see Table 2.6). Only one participant mentioned that he does not need to have an overview.

Table 2.6: Alternative techniques to maintain overview

ID	Alternative technique to maintain overview
P2	Zooming out
P3,P16	Printing and hanging the artifact on the wall
P12	Taking notes
P15	Opening the artifact twice
P17,P18,P20	Creating a higher abstraction level of the artifact
P23	Using the overview provided by the application

Discussion. The list of techniques given in Table 2.6 reveals that participants primarily use simple techniques (e.g., zooming out or opening an artifact twice) or improvised workarounds (e.g., printing or taking notes). All of these methods are not equally appropriate for all types of artifacts, e.g., only graphical artifacts can be printed and hanged on the wall. The only sophisticated

and systematic technique employed is creating a higher abstraction level. However, this is complicated and time-consuming as the participants have to do this manually.

Finding FC4. Non-software approaches are mostly used for handling the challenges of working with multiple artifacts at the same time.

Evidence for FC4. To handle the challenges related to the number of artifacts used simultaneously, 79% of the participants use multiple screens (two or three screens) and 24% of them print some of their artifacts in addition to having multiple screens (Figure 2.14).

Figure 2.14: The percentages of the methods that participants used to handle multiple artifacts at the same time

Discussion. Using multiple screens and printing increases the number of artifacts that practitioners can view simultaneously without any switching. The fact that the majority of practitioners uses multiple screens demonstrates their need for concurrently working with more than one artifact. However, both multiple screens and printing have drawbacks and limitations. The main drawback is that both exacerbate the challenge of repetitive change of focus, which is one of the main challenges we found (FB2). Moreover, the number of concurrently usable screens and printouts

is limited in three dimensions: (i) cost, (ii) available space to place screens and printouts in a work environment, and (iii) at some point searching for the needed information in a multi-screen and multi-printout environment becomes as cumbersome and difficult as keeping the same information in multiple windows on a single screen.

Finding FC5. Paper is used by all and whiteboards are used by two-thirds of the participants for creating artifacts.

Evidence for FC5. When we asked the participants whether they used any non-software ways to create, understand or manage their artifacts, we most often received "pen and paper" or "whiteboard" as an answer. Among 29 participants, 28 participants use pen and paper and 22 participants use a whiteboard.

Discussion. As discussed previously (FA5), using multiple tools to deal with artifacts has drawbacks. Software development practitioners tend to use as few tools as possible. When they decide to use a new tool (software or non-software) in addition to what they already use, it means that their current tools do not satisfy their needs completely. To find the shortcomings of the tools we asked about their reasons to use pen and paper or whiteboards, which will be discussed in FC6.

Finding FC6. The reasons for using paper and whiteboards for artifact creation include seeking more speed, flexibility and space to work.

Evidence for FC6. As the reason for why non-software tools (pen and paper or whiteboards) are used to create artifacts, the

most mentioned one was "the ease of use". We did not count this reason as a finding because being easy depends on many other factors. Instead, we looked deeper in the content of the interviews to extract the real reasons. For instance "*[P22:] It is easier to brainstorm on a whiteboard. Everyone can see it and you can erase things very quickly*" or "*[P27:] I can draw anything that is in my mind and you can find a piece of paper anywhere*". The participants mentioned that drawing on paper is easier because of the following six reasons. (i) They are not limited to the rules of a tool and they can draw whatever they want. This means that drawing on paper and whiteboards is more flexible. (ii) They have more space, especially on a whiteboard. (iii) They easily have the big picture of the artifact available. (iv) They can share it with others with no effort. (v) Paper is portable and available everywhere. (vi) They can draw faster. "Being fast" is partially dependent on other factors just like "being easy", but is partially a genuine feature of drawing on paper and whiteboards. That means, drawing on paper and whiteboards can be faster due to reasons (i)-(v) given above. Another reason could be that creating the details of a diagram (such as boxes with text) may be faster on paper or whiteboards than with a graphic tool.

Discussion. Drawing on paper and whiteboards is not necessarily a problem; it may even provide practitioners with benefits. However, when drawings on whiteboards and paper go beyond throw-away sketches, the need to be digitized at some point of time. The time and effort spent for digitizing artifacts can be saved if they are drawn in a digital tool in the first place. This can be possible if software tools provide the practitioners with the

same level of ease as paper and whiteboards. In addition to being time-consuming, the digitization process is also error-prone since practitioners may make mistakes when they want to recreate a diagram in a software tool or they may misread what is drawn on the whiteboards or paper [WSG12a].

Finding FC7. To store artifact interrelationship information, practitioners extensively use inefficient, time-consuming and error-prone methods.

Evidence for FC7. Table 2.7 shows how our participants reported to keep the relationship information of the artifacts. Only 27% of them use the ability of their software tools to keep this relationship information, and only 10% of the participants do not use any other way concurrently, i.e., they use only their software tool. According to the gathered data, practitioners mostly use folder structures to show which artifacts are related to each other (62%). In addition, 27% of the participants mentioned that they have file naming conventions in their company to show the relationships between files, e.g., by starting the name of related files with similar prefixes. 27% mentioned that they reference artifacts explicitly in other artifacts. 20% of the participants create an extra artifact that contains the information about the relationships between artifacts. Finally, more than 17% of the participants keep this information in their mind.

In the follow-up survey, we mentioned the classified methods as shown in Table 2.7 and asked the practitioners, as an open question, if they know any other method of storing relationship information.

Table 2.7: Distribution of methods of storing artifact relationship information.

Store relationship information method	Percentage
Folder structure	62%
Software tool	27%
File name convention	27%
Reference to other artifacts	27%
An extra artifact	20%
Memorizing	17%

The majority agreed that the list is complete, while some mentioned the following two methods: automatic traceability techniques based on the similarity between files, and tagging artifacts. The former is a method being researched vastly [WP10] and may be used in industry in the future, and the latter is a widely used method in different types of applications, e.g., personal task managers and social networks. Since this method is mentioned only once in the follow-up survey we will not include it in our analysis. In Table 2.7, the sum of the percentages is more than 100% since many of the practitioners use more than one notation concurrently.

Discussion. Folder structure and file name conventions can keep a limited amount of information about the artifacts and need all the members of the team to pay attention to maintain them. When referencing artifacts in other artifacts, it is not possible to have an overview of the relationships and searching is not convenient. Furthermore, this method needs maintenance whenever one of the artifacts is changed. Creating an extra artifact to keep the relationships between artifacts provides the practitioners with an easily searchable overview of the relationships. However, this needs extra effort to create and maintain. Otherwise, practitioners would

always end up with outdated and useless data. The most inefficient and error-prone approach is keeping the information in mind. The practitioners may forget and make mistakes. A new member of the team would have no understanding of the relationships in such an environment. Participant 26 brought this to the point: "*I know the connections by heart, but when someone is new it is very confusing for him*".

In addition, in the follow-up survey, we asked the participants to rate the amount of effort needed to store the relationship data in each method. As Figure 2.15 shows, the amount of effort needed to make software tools keep the relations between artifacts is much higher than that required for folder structure and file name conventions.

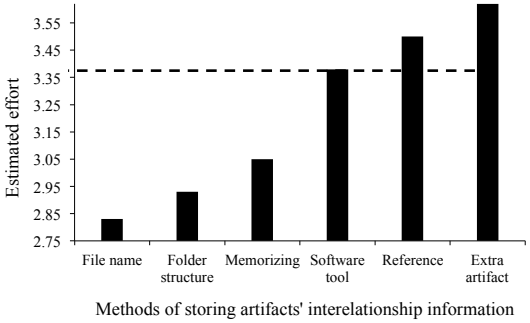


Figure 2.15: The effort needed to setup and maintain different methods of storing interrelationship information. The corresponding value for "Software tool" is emphasized by a horizontal dotted line.

This amount is very close to the amount of effort needed for referencing inside artifacts and creating an extra artifact. This

assessment justifies the practitioners' decision about which method they use. They use folder structures and file name conventions because they are easier, and use referencing inside artifacts and creating an extra artifact since they are not much more difficult. In the follow-up survey, we asked the participants to rate their overall satisfaction with the available methods to store relationships between artifacts on a five-point Likert scale. The result is presented in Figure 2.16. The majority of participants are not satisfied, with a satisfaction level in the range of "Very low" to "Medium". This shows that a decent method for storing relationship information is needed, but currently not available.

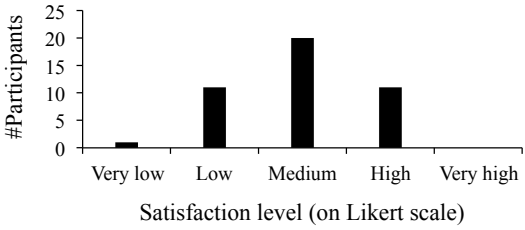


Figure 2.16: Participants' overall satisfaction with the available methods to store artifact interrelationship information on a five-point Likert scale.

2.5 Consolidation of Findings

So far, we presented the answers that we found for the first three research questions (RQ1-RQ3) in the form of findings. A summary of these findings is given in Table 2.8. Before answering the fourth

Table 2.8: Key Findings

RQ1: Properties of artifacts, screens, and tools	
FA1	Only about one-third of the graphical artifacts used by the interviewed practitioners fit on their screens.
FA2	About forty percent of the graphical artifacts do not fit on the largest screen reported in this study.
FA3	More than half of the interviewed practitioners use four or more artifacts at the same time.
FA4	More than two-thirds of the interviewed participants use customized notations for their artifacts.
FA5	On average, practitioners use more than five applications concurrently to create and manage their artifacts.
FA6	Although almost all participants work on their artifacts collaboratively, less than one-third of the collaboration is done with dedicated software development tools.
RQ2: Challenges practitioners encounter when working with artifacts	
FB1	"Relying on memory", "Searching for information", and "Maintaining the overview" are the most important challenges in handling large artifacts.
FB2	"Switching between windows" and "Working in too small windows" are the most important challenges when working with multiple artifacts.
FB3	Storing insufficient artifact relationship information provokes creating larger artifacts and makes searching and understanding artifacts more demanding.
RQ3: Dealing with the challenges	
FC1	Practitioners use their memory extensively.
FC2	Traditional zooming and scrolling are the dominating techniques for handling large artifacts.
FC3	Almost all practitioners need to have an overview of the artifacts.
FC4	Non-software approaches are mostly used for handling the challenges of working with multiple artifacts at the same time.
FC5	Paper is used by all and whiteboards are used by two-thirds of the participants for creating artifacts.
FC6	The reasons for using papers and whiteboards for artifact creation include seeking more speed, flexibility and space to work.
FC7	To store artifact interrelationship information, practitioners extensively use inefficient, time-consuming and error-prone methods.

research question in the next section, in this section, we consolidate our findings into a single comprehensive view and show how the challenges can impact usability factors.

A tool requires both sufficient diversity of features and usability to succeed. In other words, a very feature-rich software product which is not usable cannot deliver its potential benefits to the users. For example, Google Wave was a feature-rich web application which was discontinued after two years due to usage complexity [TNLJ+14]. While there is a large body of research studied ways to provide richer application tools and other researchers assessed the usability of the tools, we decided to look at the problem from a different viewpoint and find how the interaction challenges that practitioners experience in working with artifacts affect usability factors of tools. Before connecting our findings to usability factors, we need to look at the definition of usability. The ISO 9241 standard Part 11 – Guidance on Usability (ISO, 1997) defines usability as follows [Int98]: "Usability is the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use." According to this definition, there are three factors which determine the usability of a tool: effectiveness, efficiency, and satisfaction [FHH00]. *Effectiveness* is the precision and completeness with which users achieve their goals. Effectiveness can be assessed by the quality of the solution and the error rates. *Efficiency* is related to the amount of effort invested to achieve goals. It can be assessed by task completion time and learning time. *Satisfaction* is the users' comfort with which the goals are achieved. Users' satisfaction can be measured by attitude

rating scales. In Figure 2.18, we consolidate all our findings into a single view and show how they influence the three usability factors effectiveness, efficiency, and satisfaction. In order to create an expressive view, we used the pattern described in Figure 2.17. The consolidated view in Figure 2.18 consists of several chains that follow this pattern. Each chain starts with a property (▭) of artifacts, tools or environments.

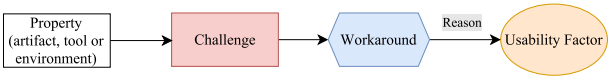


Figure 2.17: The pattern used to create the chains that are building blocks of Figure 2.18

Each property is connected to one or more challenges. Each challenge (▭) is connected to one or more workarounds (⬭) that practitioners employ to deal with that challenge. The workarounds are connected to the corresponding usability factors (⬭) that they negatively influence. Labels on influence arrows describe the rationale for the influence.

Other influence relations, which do not follow the pattern given in Figure 2.17, are depicted with dotted arrows (⇢). They may show a workaround influencing another workaround (e.g., using multiple tools causes opening more windows), a workaround affecting a property (e.g., using multiple tools makes keeping interrelationship information more difficult), or a property that intensifies another property (e.g., having larger artifacts results in partially visible artifacts).

In the subsequent paragraphs, we provide a detailed description of the chains from the six properties shown in Figure 2.18 to the

three usability factors and show how our findings (cf. Table 2.8) relate to these chains.

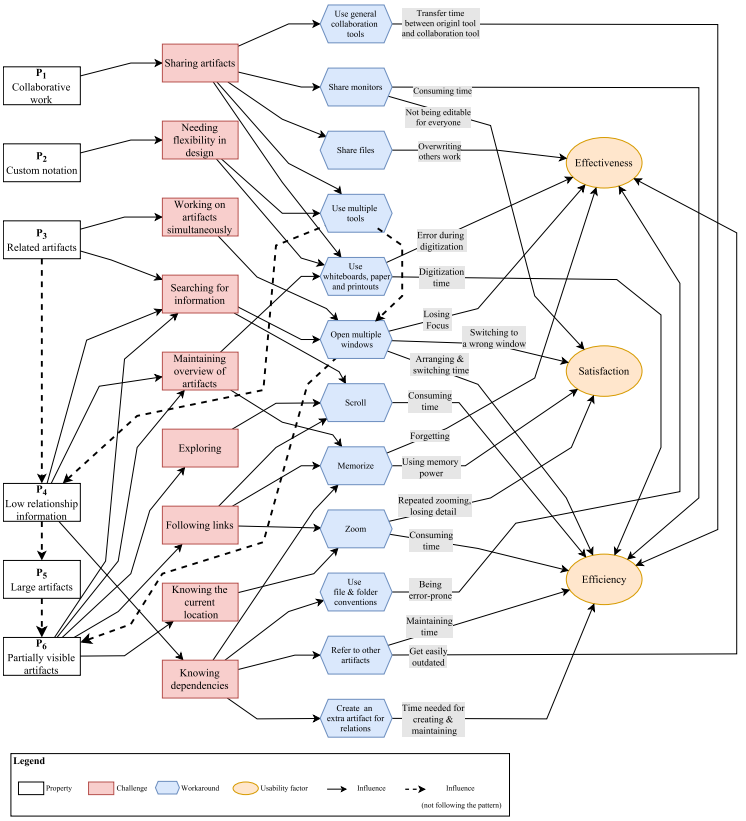


Figure 2.18: A consolidated view of all the findings of this paper. The diagram consists of several chains starting from a property and ends with a usability factor. The pattern of the chains is depicted in Figure 2.17. A full description of the chains and their relations to the findings is available in Section 2.5.

P_1 - Collaborative work. Practitioners work on their artifacts collaboratively. The first step to collaborate on an artifact is sharing the artifact. We found that the practitioners use methods such as whiteboards, paper, file sharing services and general collaboration tools since their dedicated requirements engineering or software development tools do not support all the collaboration features they require *(FA6)*. Using paper and whiteboards affects the efficiency due to the time needed for digitization, and affects effectiveness due to the errors that occur during digitization *(FC6)*. Sharing files is a quick workaround, but needs synchronization between collaborators. There is a possibility of changing something that should not be changed or be overwriting others' work. Both decrease the effectiveness of this workaround. Some practitioners share their monitors to collaborate on an artifact *(FA6)*. In this setting, only the sharing collaborator can change the artifact, while all others can only read it, which is not satisfying. In addition, only synchronous collaboration is supported and the collaborators have to dedicate their whole attention to the collaboration, which is not efficient. Another workaround for this challenge is using a specific collaboration tool (e.g., Google Docs) *(FA6)*. General collaboration tools decrease the probability of conflicting with each other. However, these tools support only a limited number of file types. Therefore, the owner of an artifact should transfer it to the collaboration tool and return it back to the original tool when the collaboration is done. This limitation makes this workaround inefficient. Generally, using a dedicated collaboration tool increases the number of tools that are being used by the practitioners concurrently, thus making them open more windows at the same time.

This workaround is discussed in P_3.

P_2 - Custom notation. We understood from the interviews that practitioners like to draw by hand *(FC5)*. One of the reasons for this is that they do not always want to comply with the notation rules *(FA4)*, especially at the beginning, when an artifact is more subject to change. This is challenging since most of the tools want their users to obey the syntax of the languages they support, while the users need flexibility. A workaround for this challenge is using paper and whiteboards *(FC6)*, which affects efficiency and effectiveness, as mentioned in P_1. Another workaround for this challenge is using the most suitable features of different tools *(FA5)*. For example, a practitioner may create one type of artifact in a tool and another type of artifact in a different tool. When using multiple tools, practitioners have to open multiple windows simultaneously *(FC4)*. We describe this workaround in P_3. Using multiple tools also intensifies the property of low interrelationship information of the artifacts (P_4) *(FC7)*.

P_3 - Related artifacts. The artifacts are related for different reasons. Due to these relationships, when a practitioner is performing a task on an artifact, specific information from other artifacts is sometimes required and the practitioners search for that information. Moreover, practitioners sometimes need to work on multiple artifacts at the same time *(FA3)*. To overcome these two challenges, practitioners open multiple artifacts in multiple windows or tabs simultaneously. When switching between these windows, the practitioners lose their focus and may commit mistakes, which affects effectiveness negatively *(FB2)*. In addition, they spend

time for managing the open windows, i.e., they arrange, hide and unhide windows according to their current task. This overhead makes their work less efficient. Also, during windows management and switching between windows, practitioners may have difficulties in finding the right window among the open windows, or they may switch to a wrong window and change something before they find out that they are in a wrong place *(FB2)*. These annoying occurrences are unsatisfactory. Opening multiple windows often cause the windows to get smaller. Smaller windows intensify the property of partially visible artifacts (P_6). Scrolling is used as a workaround for searching information in partially visible artifacts, therefore we will address it in P_6. In addition to the challenges mentioned, this property (related artifacts) makes storing information about the artifact interrelationships (P_4) more challenging *(FC7)*.

P_4 - Low interrelationship information. Absence of interrelationship information between artifacts causes three challenges. One challenge is searching and finding information (discussed in P_3 and P_6). Another challenge in such a situation is that the practitioners cannot easily create and maintain a mental image of the artifacts' relationships *(FB3)*. As a workaround, they rely on their memory *(FC1)* or print their artifacts on paper and hang them on the wall to have an overview always available *(FC3)*. Relying on memory affects effectiveness since the practitioners may forget some parts and affects their satisfaction since they need more effort to accomplish their tasks. Another challenge caused by low interrelationship information between artifacts is knowing about the dependencies between artifacts. Knowing these dependencies is crucial for understanding the artifacts and for

keeping them consistent *(FB3)*. In addition to relying on their memory, practitioners use other workarounds. They use file and folder naming conventions to show the related artifacts, which keeps a limited amount of information and needs the attention of all team members to maintain. Since the system does not enforce these conventions, making a mistake can happen frequently when employing this workaround *(FC7)*, making it ineffective. Some of the practitioners place references to other artifacts into an artifact. Since updating all these references is difficult in the case of changes, there are always some outdated references in the artifacts which make this method less effective. It is not efficient due to the time needed for maintenance. The last method for storing interrelationship information is creating an extra artifact that shows the relations. Although this method is effective, its high cost to create and maintain such extra artifacts makes it less efficient *(FC7)*. In addition to the challenges that unavailability of the interrelationships causes, we found that it discourages creating related artifacts and encourages creating large artifacts instead *(FB3)*.

P_5 - Large artifacts. We found that a considerable number of artifacts are larger than the screens on which they are displayed for viewing and editing *(FA1, FA2)*. Being large does not make working with an artifact challenging per se. However, when a large artifact is viewed on a limited-size screen, a part of the artifact always remains invisible. This problem is described in P_6.

P_6 - Partially visible artifacts. As discussed earlier in P_3 and P_5, opening multiple windows and having large artifacts result in views where only a part of an artifact is visible. This causes several

challenges. It makes searching for information and maintaining an overview of the artifacts more challenging. In addition to opening multiple windows (discussed in P_3), practitioners scroll long distances to reach the information they need, which is time-consuming and affects efficiency negatively. Also, practitioners need to explore an artifact in order to understand it *(FC2)*. Exploring becomes more challenging when a large part of an artifact is invisible *(FB1)* and practitioners scroll extensively to explore the artifact *(FC2)*. Scrolling negatively affects efficiency due to the time it consumes. Partial visibility also makes it more challenging to follow links, because most of the destinations of the links will be outside of the screen and practitioners have to scroll or go to another window to find out where the links end *(FB1)*. This is cumbersome and time-consuming, thus negatively affecting efficiency. In addition, the practitioners use the zooming workaround to follow links *(FB1)*. They zoom out to view the links and then zoom in afterward to continue their work. In addition to being time-consuming, which affects efficiency negatively, repeated zooming actions and losing detail when the artifact is scaled down are annoying and make users feel unsatisfied. Practitioners also use zooming as a workaround for the challenge of finding their current location in an artifact.

2.6 Existing Work Addressing the Challenges

In this section, we survey existing work that addresses the challenges we found in our study (see Sect. 2.4.2). Our purpose is to

explore to which extent there exist solutions to these challenges, thus providing a preliminary answer to our research question RQ4 (RQ4: Which existing solutions address the challenges of working with RE artifacts?). For definitely answering RQ4, a systematic literature review would be required which is beyond the scope of this paper. We present existing solutions in four research directions: *maintaining artifact interrelationships*, *flexibility in working with artifacts*, *collaboration*, and *navigation inside and between artifacts*.

2.6.1 Maintaining Artifact Interrelationships

Practitioners use multiple tools and each tool provides a set of functionalities and produces a set of artifacts. The artifacts are related to each other like a network. Practitioners need interrelationship information to perform different tasks. They spend a considerable amount of time, cognitive power and effort to compensate the lack of such information. In addition, considering the challenges of handling related artifacts and storing interrelationships between artifacts, a centralized view showing the relationships is necessary.

Traceability is an extensive concept in software engineering. One of its aspects is defined as the degree to which a relationship can be established between two or more products of the development process [IEE90]. Among other benefits, having a traceable software development process enables the team to know the relationship between artifacts. Numerous studies are done in the field of

traceability to maintain the traceability links from the beginning or to retrieve them when they are not kept properly. A survey in this field has been done by Winkler et al. [WP10]. The question is if these tools are being used in the software development industry. Müller et al. [MF13] found in an empirical study that the links between artifacts are often missing. They only mentioned face-to-face communication and wiki pages as the workarounds that practitioners use to find the relations between the artifacts. In another study, de Souza et al. [dSR08] found that the development teams that they studied relied on emails and instant messaging to find the change impacts. Many researchers study systematic ways of retrieving interrelationship information in the form of traceability links [WP10].

Software development tools can save practitioners' effort and facilitate the traceability link identification by storing the relationship information as the artifacts are created and evolved. For example, Codebook [BKZ10], which is a framework to connect practitioners and artifacts in a single directed graph, makes it possible to conveniently find the interrelationship information of artifacts. It keeps the relationship between artifacts and developers in a social network so that developers know whom to ask a question about the artifact they are working on.

Using one software development environment is an approach to have all artifacts in one place. However, developing software tools that address all needs related to software development artifacts is very difficult. One way to provide such a view is to have a centralized repository of artifacts. For example, IBM rational DOORS

is a requirements management system that imports artifacts of various formats. It allows the requirements engineers to create and maintain relations between parts of different artifacts. However, working on artifacts in their native applications and synchronizing them with DOORS repository requires effort. Another way to have a central management system while allowing requirements engineers to use their desired editing application is to create different small tools, instead of one large tool, such that they can be plugged into other tools. This is already done in a few tools for few features, e.g., Confluence can plug in the drawing tool Gliffy. In this case, there is a central management tool and at the same time, users can search for the pluggable tools that match their needs. Another way to achieve a centralized view would be to define a standard for artifact definition and let different tools store their artifacts in a shared repository. For example, XMI is created for UML diagrams so that different tools can store their artifacts in a common format. A common repository can be created based on a common standard way of storing artifacts.

2.6.2 Flexibility in Working with Artifacts

One of the reasons why practitioners need to use whiteboards and paper to design their artifacts is that they cannot draw exactly what they want in the tools and are limited to the templates, styles, and notations that tools provide. Furthermore, practitioners like the convenience of drawing by hand. To achieve more flexibility, tool developers should think about methods to bring the convenience

of hand drawing to their tools without turning them into mere painting tools. We are aware of two approaches into this direction. *FlexiSketch* [WSG12a], [WSG12b] looks like a free-hand drawing tool at the first glance, but it seamlessly transforms what the user draws into a diagram consisting of meaningful, related components. In addition, during this process, it lets the user define a custom notation. *Calico* [MBD+10] is a sketch-based design tool for touchscreen devices. Its focus is supporting the early stages of software design. Using Calico, practitioners can have an experience similar to paper and whiteboards when drawing and improving their designs on electronic whiteboards or tablets, while Calico enhances their experience seamlessly. In 2010 and 2011, two workshops on flexible modeling tools [OvdHS+10], [OvdHS+11] were held to investigate the reasons why practitioners prefer to work with whiteboards, paper, and other informal methods at early stages of software modeling and development, instead of using specific modeling tools.

2.6.3 Collaboration

The development of a complex software system is often a highly collaborative process, where ideas are presented and discussed by multiple stakeholders [Dek05]. The requirements engineers and software designers collaborate to create an artifact or to describe it to each other. In a distributed collaboration, the tools need to have the same features as non-collaborative versions with the additional ability to let multiple users work on a single artifact, while in a

collocated setting a larger screen is needed so that all the users can work on it simultaneously [DH07]. Various tools have been proposed for distributed software design and development. In this regard, a systematic mapping review is done by Portillo-Rodríguez et al. [PRVPB12]. IBM Jazz is an extensive collaboration platform that covers various areas of a software engineer. Rational Requirements Composer is a tool based on this platform and lets requirements engineering collaboratively work on requirements artifacts. A hybrid approach to solve the collaboration problem is proposed by Wüest et al. [WSG15] in which each collaborator has its own device (tablet) while a large screen and a large electronic whiteboard is their common point of reference.

General collaboration tools allow users to edit even the same sentence simultaneously without interrupting each others' work. For this, Google Docs is an example. Software development tools should incorporate these successful collaboration practices so that practitioners can easily collaborate without leaving their software development environment. Such features encourage collaboration, improve awareness and save time. There are some software development tools that allow collaborative work, such as letting users have shared Scrum boards (e.g., Trello) or edit their wiki pages collaboratively (e.g., Confluence). However, they do not support all types of artifacts and all types of collaboration.

2.6.4 Navigation Inside and Between Artifacts

Practitioners spend a considerable amount of time using traditional navigation mechanisms such as scrolling, zooming, and opening

multiple windows. They make mistakes and perform unsatisfactory actions while using these mechanisms. All of these affect their productivity negatively. In addition, screen space for displaying information has always been limited. Therefore, the way users interact with the information displayed on the screen has been the focus of a large number of studies. These studies aimed especially at optimizing the presentation of information. Two different approaches can be taken for this purpose: increasing the screen space or utilizing the available screen space more efficiently. In the first approach, using large screens or arrays of multiple screens is being investigated. Czerwinski et al. [CSR+03] conducted an empirical study to examine the productivity benefits of larger display screens and found a significant performance advantage. Lischke et al. [LMW+15] used multiple monitors to have a wall-size screen in an empirical study and measured the task completion time in different settings. They reported that the optimal monitor number is three.

In the second approach, techniques such as zooming, overview + detail, focus + context and cue-based methods are employed to display as much useful information as possible on the screen [CKB09]. Lam et al. [LM10] analyzed 22 studies that implemented such techniques to extract design guidelines indicating when and how each of these techniques should be used. Generally, tools should use the space of the screen more intelligently, show the needed information only, gather information from multiple sources into a single view, e.g., by employing semantic zooming and different levels of detail [KM07]. Additionally, tools can reduce the amount of information that practitioners keep in their mind about what

exists outside of the screen, e.g., by providing such information on the border of the screen [FD10].

In this direction, initial steps are taken in some tools. However, what is actually implemented remains far behind the state of research. Some tools show an overview of the artifact in a small window somewhere on the screen which helps the user know where the working region is located in the artifact and lets the user navigate faster. Although this is an old way of dealing with complexity, few tools have incorporated such a feature. As another example, the Fisheye concept [Fur86] has become a standard for source code version control. In such tools, the changed line is highlighted and is shown together with a few lines before and after it. The rest of the source code is hidden and represented by a symbol such as three dots. In a conceptual solution, FlexiView [GSG15] has extended the Fisheye concept to other aspects of artifact navigation to accommodate only the information on the screen that is required for the current task. These tools reduce the zooming and scrolling actions and show information from different sources in a single view to decrease the number of concurrently open windows.

However, both of these approaches give rise to other challenges such as arranging the windows and tracking the mouse pointer [RCB+05]. Therefore, for improving the performance of existing user interfaces, it is not sufficient to just increase the screen size or employ a smart visualization mechanism. Instead, the design of user interfaces needs adaptation, which requires understanding the new challenges. Furthermore, the findings of these researchers depend on the information type (e.g., graph or 3D model), the interaction

type (e.g., comprehension or manipulation) and the users. Consequently, the provided guidelines need to be tailored to requirements engineering.

Requirements visualization is another broad area of research that investigates how graphical models of requirements should be created [CJLGG09]. However, the research in this area does not address how the created graphical models should be presented to users. For example, Cleland-Huang et al. [CHH07] proposed visualization techniques such as hierarchy structures to enhance the understandability of artifacts in automatic tracing tools. Reddivari et al. [RCN12] designed a tool to support the exploration of requirements via quantitative visualizations. The true benefits of these tools will not be realized unless the artifacts are presented to users in the most effective form. For instance, Reinhard et al. [RMG07] [RMS+08] developed a custom-made presentation technique to fully exploit the potential of the requirements modeling language Adora.

In addition to enhancing the understandability of the artifacts, software engineering tools can provide cognitive support [Par13b]. If a complex diagram is not presented hierarchically, viewers have to derive the hierarchy in their mind [KDvV02]. Cornelissen et al. [CVDMZ07] established a set of metrics for scenario diagrams to recommend a number of abstractions that should be used to have the desired amount of detail. Bennett et al. [BMS+08] reported the usefulness of their interaction features for sequence diagram navigation. Presenting information in a clear pattern helps to remember the relationships [KDvV02]. In a data-intensive field like requirements engineering, offloading some of the cognitive load

is a requirement for any tool which supports viewing and editing of artifacts.

2.7 Related Work

In the previous section, we discussed a number of related studies that addressed the same challenges we found in our study. In this section, we discuss related works that study RE practitioners and how they work with RE artifacts and how they use tools that support them.

Software engineering heavily involves people. In order to understand software engineering and enhance different aspects of it such as working with RE artifacts, the first step for researchers is to study people empirically. In an empirical study, based on the selected research method, a suitable data collection technique should be chosen as well [ESSD08]. Lethbridge et al. [LSS05] compiled a list of various techniques of acquiring information about people and how they work in software engineering environments. They have cited successful prior experiences of such studies and applying their findings to improve software engineering. For example, as data collection techniques, they have used interviews and surveys in another work [LSF03] to study how software engineers use documentation. They reported that documentation is most of the time outdated and documentation is often poorly written. However, their study did not seek for the reasons.

Empirical studies in RE investigate various aspects of RE such as the effects of improved RE over an entire project lifecycle [DC06], user involvement in software engineering [PB13], and effectiveness of elicitation techniques [DDH+06]. There are few empirical studies of requirements artifacts and how they are being used by practitioners. Liskin [Lis15] has interviewed practitioners to understand how they manage to work with multiple artifacts and how they link their artifacts together. Winkler [Win07] has performed a survey to find which artifacts are created in the requirements phase of software engineering and how information flows between related artifacts. The last two mentioned studies consider the artifacts and their usage without considering the tools that make using artifacts possible. In contrast, we look for the interaction challenges of working with artifacts using tools.

RE tools have been evolving over time to meet the demand for flexibility, agile development, collaboration, and new ways of requirements management. De Gea et al. [dGNA+11] carried out a survey to identify the existing RE tools and compared them feature-wise. In spite of many existing specialized RE tools, Forward et al. [FL02] reported that most preferred tools for creating artifacts include word processors and text editors. Limited investigation has been done on the reasons why RE tools are not used frequently. Karlsson et al. [KDoD+02] reported that one of the small companies that they have studied requested a simple RE tool since the existing ones would have a too large introduction overhead and a too steep learning curve. Hoffmann et al. [HKWB04] created a list of requirements of RE tools. The only proposed requirement

in their work that addresses a challenge of our study is the ability to work collaboratively.

Almost 17 years ago, Myers et al. [MHP00] predicted that a radical change in user interface design would occur. While we actually observed that UIs have matured since then, fundamental changes did not happen, especially in the tools that are used for requirements artifacts. Consequently, there is room for improvement in the UI of RE tools. Based on the related works that we reviewed about empirical studies on requirements artifacts and tools, there is no clear evidence in the literature to what extent working with RE artifacts is challenging and how successfully RE tools employed interaction techniques to enhance working with artifacts including creating, manipulating and managing artifacts. Our study contributes to filling this knowledge gap, thus providing an empirical basis for identifying the interaction requirements of RE tools.

2.8 Threats to Validity

We identified the threats to the validity of our research and tried to minimize their effect on the final result. Below we discuss the usual four categories of validity [WRH+00].

Conclusion validity refers to finding a relation between data if it exists. Measurement reliability can affect conclusion validity. Therefore, to make our measures clear for the participants, we described our measures in detail and in a step-by-step manner. In

addition, all interviews were conducted by the first author. When we needed a new measure, we defined it by combining other well-known measures (e.g., screen size and artifact size). We verified the consistency of our measures by asking duplicate questions. Moreover, we discussed the questions with RE experts and did a pilot study to avoid misunderstandings. Since we asked the participants to imagine having a screen size that they did not have in reality, still the accuracy of the gathered data depends on how accurately they can imagine that situation.

We made assumptions about what kind of people are more likely to have experienced challenges. Different assumptions might have resulted in different orders in our lists. Furthermore, we tried to have the interviews and surveys in similar conditions for the participants. We suggested the participants spend around ten minutes for the survey. We scheduled the interview meetings in advance and asked the participants to be in a non-disruptive environment. Using statistical computations can be a threat to the validity of the results especially on lower sample sizes. In this study, we used such computations for prioritizing the challenges and not for identifying or filtering the challenges. In addition, we did not distinguish the practitioners who wrote artifacts and the practitioners who only read artifacts. This factor can affect the prioritization of the challenges.

Internal validity of an interview refers to making sure that the differences in the answers received are only because of the known differences among participants. Questions remained the same during the whole duration of the study. All 29 interviews were

performed within a relatively short period of two months to avoid any software or hardware technology advancement. All participants were self-motivated and we did not offer any compensation.

Construct validity ensures that questions actually ask what they are supposed to ask. For example, we cannot guarantee that the participants remember everything related to our questions during the interview. So if a participant did not mention a challenge, this does not necessarily mean that they did not face that challenge. Therefore, in the analysis phase, we tried to minimize the influence of the frequency of the answers by prioritizing the challenges based on how much the participants were challenged. Moreover, although we did not have any hypothesis or expectation about the results, we were careful not to let the participants guess any hypothesis or expectation by mistake. For instance, we chose neutral tone in the interviews and ordered the questions of the survey randomly.

Our goal in this study was to investigate how practitioners work with RE artifacts, the challenges they face, and the related properties of artifacts. Eventually, we studied software engineering and RE artifacts together. The reason was that the challenges of working with requirements artifacts are not confined to RE, and recruiting participants that purely work with RE artifacts or can isolate their experience with RE artifacts from other artifacts was not possible. To assess this threat, later in the study, we compared the results from the participants who worked mostly with RE artifacts and the results from the participants who worked with more software engineering artifacts. The comparison did not show any statistically significant difference that we could report as a

finding. Therefore, we conclude that our findings are valid for RE artifacts.

There are two threats concerning the ability of zooming. First, we did not investigate how much of an artifact is needed for a given task. It is possible that only a part of a large artifact is sufficient for carrying out a task. In this case, despite having an artifact that does not fit the screen, some of the challenges that we found may not apply. However, even in such a case, the practitioner has to scroll to find the needed part and adjust the zoom level to make it fit the screen. Second, we assumed that the resolutions of participants' screens are high enough to show the details of an artifact adequately when zoomed out extensively and the limitation for zooming out without making details unreadable is imposed by human eyesight only, not by the resolution of the screens. We made this assumption to reduce the duration and complexity of the interviews, considering that the probability of practitioners using outdated hardware is not high. In addition, to increase the accuracy of the measurement and make the participants more relaxed, we informed them that the data will be used and presented anonymously. We gathered information from various sources to avoid mono-operation bias.

External validity of a research means that the results are generalizable. For this purpose, the selected sample (the interviewees in our case) should not have certain features in common. This is very hard to achieve in an interview-based study. Some features were inevitably shared by all participants such as being volunteers who are interested in contributing to a scientific study and are social enough

to answer our e-mail and participate in a one-hour long interview. To avoid bias, we defined our criteria for selecting participants as simple as possible and used two different types of sampling. The variety of our final sample in terms of country, roles and company size shows that we were successful. Nevertheless, we cannot claim that our sample of 29 practitioners is statistically representative of the whole software development community. Consequently, as stated at the beginning of Section 2.4, the quantitative figures we report in our findings are not statistically generalizable. Hence, with respect to generalizability, our quantitative findings should be considered as hypotheses, rather than generally valid facts.

2.9 Conclusion and Future Work

In this paper, we presented a study about working with requirements and software development artifacts. We considered the properties of artifacts and tools that are related to information presentation and interaction. We also investigated the challenges related to these properties and the workarounds that practitioners employ to overcome them. Our goal was to gain an in-depth understanding of the state of practice in this area. To achieve this goal, we interviewed 29 practitioners from different companies located in eleven countries.

Our findings clarify the relations between the mentioned properties of artifacts, the challenges related to them, and how these challenges are handled in practice. We found that practitioners

work with artifacts that are larger than their screens and with interconnected artifacts that have to be accessed simultaneously. In addition, they need to collaboratively work on their artifacts and keep the relationship information of the artifacts. Since the existing software tools do not provide sufficient support for conveniently carrying out such tasks, the practitioners we interviewed try to address the challenges encountered in various ways. For instance, they heavily rely on their memory or use other methods that are inefficient and frequently error-prone, e.g., taking screenshots to remember, using paper and whiteboards to create artifacts, and using file name conventions to track the relations between artifacts.

Furthermore, our analysis of how our findings relate to other studies that address the challenges we identified encourages researchers and tool developers to study the reasons why these challenges exist in spite of the existing solutions.

Requirements engineers are strongly involved in creating and managing artifacts. So when framing our work in the context of RE, we can state that our results contribute to a better understanding of the challenges that requirements engineers face in working with requirements artifacts. Addressing these challenges is one way of improving RE tools. More efficient and more effective tools will enable requirements engineers to work with artifacts with less effort.

As a next step, we plan to use the results of this study to develop new approaches that enable practitioners to handle challenging

artifact types, such as large artifacts or concurrently used sets of interconnected artifacts, in an efficient and effective way. We also envisage further studies focusing on individual challenges or a group of related challenges to discover specific requirements for future tools, e.g., by observing a team of requirements engineers when they collaboratively work on artifacts or inspecting the artifact interrelationship information they keep and store.

2.10 Acknowledgements

The authors thank Dr. Irina Todoran Koitz for her insightful discussions and valuable feedback. We are grateful to the anonymous reviewers for helpful suggestions that enabled us to improve this paper. We would also like to thank all the participants and their companies for their time and valuable contributions that made the data collection possible for this research.

Chapter 3

A Physics-Based Focus+Context Presentation and Navigation Technique for Requirements Artifacts

Original publication:
FlexiView: A Magnet-Based Approach for Visualizing Requirements Artifacts
P. Ghazi, N. Seyff and M. Glinz
International Requirements Engineering Conference on Foundation for Software Quality 2015

Abstract

Requirements engineers create large numbers of artifacts when eliciting and documenting requirements. They need to navigate

through these artifacts and display information details at points of interest for reviewing or editing information. Traditional visualization mechanisms such as scrolling and opening multiple windows lose context when navigating and can be cumbersome to use, hence. On the other hand, focus+context approaches can display details in context, but they distort the data shown (e.g., fisheye views) or result in a large display canvas which again requires scrolling (e.g., zooming in ADORA). We are developing a novel method for displaying just the information needed to perform an intended task. Our method partitions the available screen space into regions. The boundaries of regions are simulated with a model consisting of virtual magnetic balls and springs that behaves like a physical system. This model supports the requirements engineer in selecting how the relevant information should be displayed. In this paper, we present preliminary results on how our conceptual solution works and what benefits are expected.

3.1 Introduction

When eliciting and documenting requirements, requirements engineers create a large number of artifacts (e.g., documents, models, or sketches). Creating and working with these artifacts on electronic devices entails two visualization problems, particularly when working with displays of limited size (e.g., tablets): (i) There are artifacts such as large models or sketches that are larger than the available display. (ii) A requirements engineer frequently needs to view more than one artifact concurrently in order to comprehend or edit these artifacts. Todays tools employ traditional techniques for tackling these visualization problems: the first problem is typically addressed by scrolling and the second one by opening multiple windows [CKB09]. These techniques work well for focusing on individual pieces of information, but they do this at the expense of losing the information about the context that those pieces are embedded in. Therefore, working with traditional visualization mechanisms is cumbersome when the elements to be displayed in detail are part of a network of interconnected elements, which is typically the case in Requirements Engineering (RE). On the other hand, there are so-called focus+context visualization approaches that can display details in context [CKB09], [KM07]. However, the existing approaches distort the data shown (e.g., fisheye views) [Fur86], [SB94] or result in a large display canvas which requires scrolling (e.g., zooming in ADORA) [RMS+08]. In our research, we are developing a new visualization mechanism called FlexiView which solves, in a unified way, both visualization problems mentioned above. Based on a physical metaphor of

magnets and springs [Ead84], [RK14], [SF08], FlexiView shall be able to flexibly visualize detailed requirements artifacts without losing the surrounding context within a display canvas of fixed size. In contrast to existing visualization mechanisms, FlexiView will be designed such that it can be used for visualizing both single artifacts (e.g., a graphics model diagram or a sketch) and a network of multiple different artifacts. The rest of the paper is organized as follows. In Section 3.2, we briefly discuss the goals of our approach. Section 3.3 reviews the relevant literature. In Section 3.4, we present our approach and discuss its features and benefits. Section 3.5 concludes the paper

3.2 Research Goals

Our goal is to develop a unified focus+context visualization mechanism which is tailored to requirements engineering. With our approach, we aim at overcoming the problems of existing visualization approaches for RE artifacts, thus allowing the construction of innovative RE tools (e.g., for supporting lightweight requirements modeling [Gli10]) as well as improving the way how existing RE tools visualize information. We envisage that such tools will (i) reduce the time and energy spent on navigating among various artifacts, (ii) prevent users (requirements engineers as well as stakeholders and developers) from getting lost in the navigation space, and (iii) make the set of RE artifacts better comprehensible for users. We expect that our visualization mechanisms will be useful also for visualizing other artifacts, e.g., in software architecture, but we will concentrate on RE artifacts in our research.

3.3 Related Work

Scrolling and opening multiple windows are traditional ways to deal with a large number of artifacts. They have been used in almost all available user interfaces. However, they lose context and create visual discontinuities, thus causing cognitive over-head for the user [CKB09]. Focus+context visualization techniques display the focus within its context in a single continuous view. The theoretical foundation for focus+context interfaces was established by Furnas [Fur86], who describes generalized fisheye views. This is a general interaction framework for information filtering according to the users current point of interest. This concept was later used for creating Graphical Fisheye Views (GFV) [SB94]. GFV is a non-linear distortion-oriented graphical visualization technique and supports multiple foci. The results are sometimes reported as too distorted. Many derivations of fisheye views can be found in literature, such as JellyLens [PPCP12], that morphs around arbitrary geometric features in the data. In the ADORA project, a fisheye zoom algorithm for visualizing and manipulating hierarchical graphical ADORA models was developed [RMS+08]. The algorithm provides an editable layout which is stable under multiple zooming operations. However, zooming in multiple points may result in a large canvas which requires the user to scroll again. In the field of graph visualization, many techniques and algorithms have been created for viewing large graphs. A particular thread of work deals with manipulating graph visualizations based on a physical metaphor [RK14], treating graph nodes as metal balls and edges as springs that are flexibly attached to those balls [Ead84].

By applying forces to such a network of balls and springs, for example by placing magnets, interesting parts of a graph can be highlighted or magnified [SF08], thus allowing the construction of intuitive, user-friendly graph visualization and navigation mechanisms.

3.4 FlexiView: A Magnet-Based Visualization Approach

FlexiView combines the concepts of fisheye zooming and magnet-based graph visualization into a new technique for visualizing and manipulating requirements artifacts. We have chosen this technique due to its potential for solving both visualization problems mentioned in Section 3.1 (visualizing large individual artifacts as well as sets of interconnected artifacts) in a uniform way on display devices of limited size. Subsequently, we illustrate the idea using a typical scenario occurring in early stages of requirements engineering: we have a set of interconnected artifacts, each artifact being a chunk of text, a sketch, a model fragment, an image, etc.

3.4.1 Conceptual Solution

FlexiView partitions the whole working space into regions in such a way that each region contains just one element (i.e., a single artifact in the scenario mentioned above). For the sake of simplicity, we will

call these elements objects. Unlike other visualization techniques, users interact with regions instead of objects. The interactions of the users affect the regions and any change in the regions affects the objects consequently. To manipulate the size of the regions, we model the region boundaries with a physical spring model [Ead84] (Fig. 3.1a) and apply forces to that model using virtual magnets [SF08] (Fig. 3.1b). The four balls in the corners of the display space are considered to be fixed and neutral. All other balls can move and are considered to be magnetic, having a negative pole on their surface. The balls positioned on a horizontal or vertical edge of the display space can only move horizontally or vertically, respectively. The other balls can move in any direction. In its initial position, the model is in balance. Users can now manipulate the size of regions by creating virtual magnets anywhere on the screen. These magnets have a single pole on their surface. The position, strength and polarity of these magnets determine how the regions change: any magnet repels the balls of the same polarity and attracts the balls of the opposite polarity. The placement of virtual magnets on the drawing space applies forces to the movable balls and makes them move, thus compressing or stretching the springs attached to the balls. Springs apply forces to the balls in return. The balls move until the forces of springs and the magnet(s) applied to them neutralize each other. The system is in balance again until the user changes the layout by creating or removing a magnet, moving an existing magnet, or altering its strength. Creating multiple magnets affects multiple regions simultaneously.

Figure 1b shows the forces and the resulting re-positioning of balls

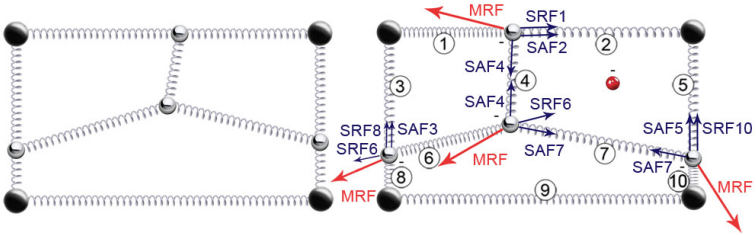


Figure 3.1: (a) A sample of regions modeled by metal balls and springs. (b) The positions of the balls are determined by three forces: the Spring Repulsive Force (SRF), the Spring Attractive Force (SAF), and the Magnet Repulsive Force (MRF).

when a magnet of negative polarity is placed in the top right region. Compressed springs apply repulsive forces and stretched springs apply attractive forces. The directions of spring forces are aligned with the directions of the springs. The direction of the force that the magnet applies is given by the straight line between the magnet and the ball. The balls on the boundaries of the drawing space behave in a restricted way as described above. The size and the position of the objects are controlled by the regions they reside in. When the position or the size of a region is changed by the user, the new position or size of the objects residing in that region will be calculated accordingly. The result will be the enlargement or shrinkage of objects. Eventually, a new view of the original information is produced. Figure 3.2a shows three steps of a user interaction. The first image (3.2a) shows some objects representing requirements artifacts, their relations, and their regions. The regions are modeled by our balls and springs model. In the second image (3.2b), the user has created a

virtual magnet with negative polarity (the red ball) in the region of interest. The magnet has repelled the balls and caused the region of interest to increase in size. The object in this region is enlarged and can be displayed in more detail, hence. Conversely, the bottom left region has become too small to display its object in detail, so this object is replaced by a more abstract representation. In Figure 3.2c the user has increased the power of the magnet, resulting in a larger region of interest and further shrinkage of the other regions.

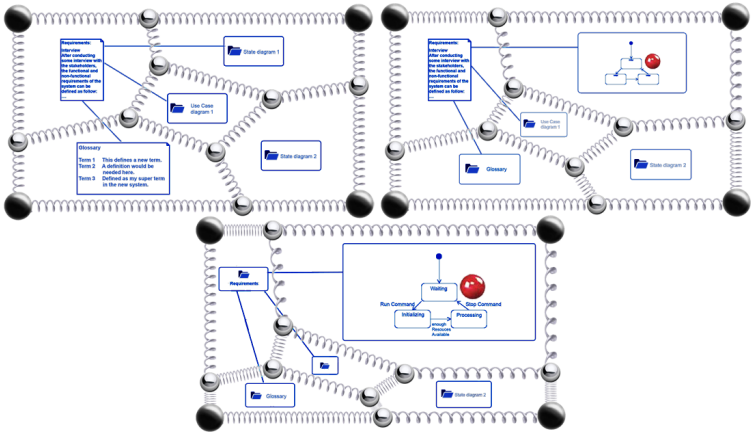


Figure 3.2: (a) RE artifacts and their regions. (b) The user has placed a magnet in the top right region, resulting in the enlargement of this region and the appearance of more details. In the shrunk region at the bottom left, fewer details are displayed. (c) The user has increased the strength of the magnet, so the corresponding region grows and the other ones shrink.

In order to replace objects in shrunk regions with more abstract

representations and those in enlarged regions with more detailed ones, we keep display metadata for all objects [FDB08]. We assume that we have at least a three-level hierarchy: project artifact contents of an artifact. If an artifact, for example, is a symbol-and-line drawing, the symbols in that drawing constitute another level of a hierarchy. The applications used to create and edit artifacts store them in their own file format on local or remote storages or in repositories. We assume that these applications provide a kind of plug-in of FlexiView such that FlexiView can access the information required to display the artifacts and/or their constituents. Thus, users can explore information by navigating in and between artifacts with FlexiView while they can still manipulate and modify the content shown using the corresponding applications.

3.4.2 Algorithms for FlexiView

We are currently exploring existing graph manipulation algorithms that can be adapted for implementing the FlexiView approach. As in other work [SF08], we do not strive for physical accuracy, modeling exactly Hookes law for the springs and the laws of magnetism for the magnets, but use the physical model as a metaphor for guiding algorithm design. The users of FlexiView will not have to bother with physics. For them, using a magnet will feel like having a wizard that magnifies the region of interest on the display by a user-controlled factor and shrinks the rest accordingly.

3.4.3 Expected Benefits

Keeping the overview. A strong magnet can enlarge a region up to almost the whole working space and consequently shrink the other regions and their residing objects down to almost a dot. However, the overview still exists. Although the undersized objects may be unclear, showing their relations and their positions keep the complete image of the information in the users mind.

Minimizing distortion. All focus+context techniques distort the image of the information. In FlexiView the information inside each region alone is not distorted. The overall distortion available gradually increases when moving away from the current foci and decreases reaching far regions. Furthermore, the neighboring structure and relative position of the regions is kept intact. This way, the user is still capable of mapping the produced view to the original one, thus causing less disorientation.

Editing ability. Distorted views may improve the visualization, but are not pleasant when it comes to editing tasks. In our approach, each region acts as an undistorted drawing canvas which enables users to edit information conveniently.

Being reversible. The altered views of the information are temporary views which are produced during specific tasks. The benefit of using magnets as tools of interaction is that by removing them, the original view reappears on the screen immediately. Moreover, the sequence of creating magnets on the screen can be undone not only in the reverse order but any magnet can be removed regardless of existing magnets created after it.

3.4.4 An Application Scenario

We illustrate the expected benefits of FlexiView with an application scenario from RE. Imagine a requirements engineer works on a requirements change request concerning the behavior of component X. Lets follow this engineers work through a sequence of steps. (1) The engineer starts from an overview that displays an interconnected set of requirements artifacts. (2) She places a magnet on the component X icon so that the constituents of component X appear. (3) She then places the magnet on the state machine icon of component X and increases the strength of the magnet until the state diagram appears. (4) Now she can study this diagram and figure out how it would be impacted by the requested change. (5) Next, she wants to know the corresponding stakeholders. Placing another magnet on the pre-tracing link, she follows that link to the list of stakeholders, where she intensifies the strength of this magnet to see the actual stakeholders for the state machine of component X (the size of the state machine will shrink when displaying the stakeholder list, but it will remain a focus area on the display as its magnet is still there). (6) For a critical stakeholder, the engineer now wants to view this stakeholders business goals. She moves the second magnet from the stakeholder list to the business goal specification, following the corresponding link. The stakeholder list disappears as soon as the magnet is moved and the region containing the business goal specification is enlarged. (7) By controlling the intensity of the magnet, she can now navigate into the business goals. (8) Having studied this information, she now wants to modify the state machine of component X. As the

magnet on the state machine of component X is still there, she just removes the magnet from the business goal specification and the display reverts exactly to the situation that she had in step (4), thus allowing her to make the intended modification.

3.4.5 Research Status

We started this research in spring 2014 with conducting a thorough literature review. Based on the results of this review as well as an analysis of navigation and visualization problems identified in our FlexiSketch project [WSG13], we have developed the concepts of FlexiView as a new technique for visualizing and manipulating requirements artifacts. We are currently investigating algorithms for implementing our approach. Our research will continue with actually implementing FlexiView and creating a test environment which will allow us to evaluate our approach against other approaches for visualizing and editing a set of requirements artifacts. We will evaluate the usefulness of FlexiView for performing typical RE tasks such as creating and understanding artifacts, tracing and change management. Additionally, we will deploy our approach on FlexiSketch [WSG13], where we plan to conduct real-world evaluation studies.

3.5 Conclusions

In this paper we have previewed FlexiView: a novel visualization technique which aims at enabling requirements engineers to work

with multiple interconnected artifacts on screens of limited size and, using the very same visualization technique, enabling them to navigate in artifacts that are larger than the available screen. Based on its underlying physical metaphor of springs and magnets, we expect FlexiView to provide seamless and natural looking multi-focus zooming. Due to its generic nature, FlexiView will be embeddable in both existing and novel tools that manipulate requirements artifacts such that these tools deliver their services through the FlexiView visualization mechanisms.

Chapter 4

An Imitating Graph for Faster Usability Tests of Requirements Modeling Tools

Original publication:
ImitGraphs: Towards Faster Usability Tests of Graphical Model Manipulation Techniques
P. Ghazi and M. Glinz
International Workshop on Modelling in Software Engineering (MiSE@ICSE) 2017

Abstract

Due to the increasing use of both general-purpose and domain-specific graphical models (e.g., UML diagrams or graphic DSLs) in

different stages of software development, software engineers who work with these models spend more time interacting with modeling tools. Thus, the usability of the interaction techniques employed by modeling tools affects the overall productivity of software development. Tool developers and user interface designers rely on the feedback from usability tests to optimize the user interface of tools that provide a graphical editor. Developing a working prototype to test new techniques is costly due to the complexity and variety of graphical models. This results in either tests at the late stages of development when changes are more expensive, or tests with prototypes that only support a subset of the intended graphical models. In order to simplify conducting usability tests, instead of using the intended graphical models in the tests, we propose to use simpler models that require similar interactions when being manipulated. For this purpose, we introduce graphs with additional properties, which we call ImitGraphs. ImitGraphs can be parametrized such that their interaction behavior is similar to that of an intended graphical model. Further, we introduce a method to instruct test participants to create ImitGraphs and manipulate them. ImitGraphs enable tool builders to develop prototypes for usability tests faster and consequently cheaper, thus resulting in more usability tests at early stages of tool development and on a wider range of intended models.

4.1 Introduction

Graphical models are used in various stages of software development, such as requirements engineering, software design, implementation, test, and maintenance. Therefore, the need for an effective and efficient user interface for manipulating these models is eminent. Designing a new user interface technique with a high usability is an iterative process [May99], which requires multiple cycles of testing and improving. In such tests, participants are asked to perform predefined tasks using traditional and new techniques, giving the developers the chance to measure the improvement their techniques bring. Ideally, usability tests should cover all of the prospective graphical models [Lew06]. However, since conducting such tests is expensive, designers either test a subset of their intended graphical models or test at a late stage of tool development.

The high cost of usability tests is due to the complexity and variety of graphical models. Thus, implementing a working prototype that handles the intended complex models becomes a demanding task. Eventually, user interface designers end up with complex code for their prototypes, which is difficult to change after receiving feedback from the tests. The possibility of widespread changes being required makes tool developers unwilling to invest in a comprehensive prototype. Instead, they prefer to test their interaction techniques on a specific type of graphical model [FD10, Eig03, Eic08, Sch11].

To circumvent the complexity of graphical models, their equivalent simple graphs are already used for layout optimization purposes [Spö15], but not for user interface usability tests. Conducting user interface usability tests would be much easier, if they could be conducted on graphs instead of the graphical models. In that case, tool developers could implement new techniques very quickly for simple graphs, resulting in lower cost or more tests, and would allow their techniques to mature. However, this is not possible due to the high degree of simplicity in graphs compared with graphical models. The simple nature of graphs results in needing simpler manipulation techniques. Therefore, an optimized user interface for graphs is not necessarily appropriate for graphical models and similarly, the feedback from usability tests on simple graphs is not completely valid for complex graphical models. Inspired by this idea, we asked this question: *"Can we define a special type of graph that is simple but has enough complexity to model graphical models?"* If such a type of graph can be defined, usability testers can use it in user interface usability tests resulting in a faster implementation of a working prototype and a higher number of testing iterations. Finally, after achieving an effective and efficient user interface, they can implement it for the original graphical models.

In this research, our goal was to study graphical models used in software development and find a way to define a special type of graph that can be used in usability tests of manipulation techniques instead of graphical models. To achieve this goal, we studied the process of manipulating diagrams and the interaction steps that a modeler takes. We found why simple graphs behave differently

compared to their original complex graphical models. Based on these findings we defined an extended type of graph that can be specialized to imitate the behavior of graphical models in tests and named it ImitGraph.

Our contributions are: (i) the definition of ImitGraphs, (ii) a method to specialize them, (iii) a set of commands to instruct participants of usability tests to draw ImitGraphs similar to the way they draw graphical models.

4.2 Related Work

Studies on human-computer interactions have provided principles of designing user interfaces with high usability [Shn10]. However, even after following their guidelines the usability of the software product needs to be evaluated and improved [May99]. Usability testing is a fundamental way of usability evaluation [Lew06]. When usability evaluation is carried out at a late stage, changes to the interface can be costly and difficult to implement [Hol05]. Therefore, various methods for acquiring early feedback are proposed, e.g., rapid prototyping techniques [Nie94]. Although the goal of our study, which is making early usability tests possible, is similar to those methods, we exploit the properties of the graphical models that our targeted modeling tools should handle.

Most of the research on enhancing the user interface of modeling tools focused on the visualization, navigation and rarely on

manipulation techniques, e.g., onion graphs for visualizing UML class diagrams [KM07], semantic zooming for navigating UML diagrams [FDB08] and off-screen visualization technique for UML class diagrams [FD10]. Despite these efforts, working with user interfaces of modeling tools is still considered as arduous [MAB+14].

Another group of studies enhanced the visualization by optimizing the layout of graphical models using graph layout algorithms [Spö15]. In these works, the information about the elements of the graphical models is transferred to graph layout libraries along with settings and configurations, and these libraries return an optimized layout. Graphviz [EGK+04] is a popular library in this field. In addition, there are other works that created dedicated algorithms for automatic layout of specific graphical models such as class diagrams [Eig03], use case diagrams [Eic08] and data flow diagrams [Sch11].

Most authors of related works have evaluated their proposed approaches on one type of graphical model only due to the expensive evaluation experiments. In this study, our goal is to benefit from the simplicity of graphs not only for layout optimization but to make usability tests faster on a wide range of graphical models without the need for an ad-hoc transformation implementation.

4.3 Graphical Models and Graphs

Evaluating new interaction techniques requires testing experiments which are costly. One of the main reasons of their high cost is

the complexity of the graphical models for which a working prototype should be created. Therefore, we set our goal to find a suitable substitute to represent graphical models in such experiments. Specifically, our goal was to find a model with the following properties: (i) imitate the behavior of graphical models when being manipulated, (ii) be simple enough to allow quick implementation of new interaction techniques, (iii) has the potential to represent a large group of graphical models, and (iv) be easy to learn for the participants.

Since a simple graph already meets the requirements (ii)-(iv), it is a good starting point. We will discuss the remaining requirement, i.e. its behavior, in the rest of this section.

4.3.1 Diagram Manipulation in an Experiment

In order to explain a sample behavior of graphical models during manipulation, we made the following hypothetical scenario, in which the participant of a test is given the description of a process and is asked to draw a UML activity diagram. The following process describes how the system issues an invoice in an online shop: *The system first receives an order from a user, then, it issues the invoice. Before issuing the invoice, if the user is a member, the system applies a discount. In parallel to checking the user's membership and applying the discount, the system estimates the delivery date to be included in the invoice.*
In our hypothetical scenario, the participant reads the process description and models the acquired information in an activity

diagram as he goes on. After reading each part of the description and analyzing it, he applies the required changes in his mind. At certain points, he decides to transfer the changes from his mind to the actual diagram. This scenario is presented in Figure 4.1. The first column shows the description and is separated at the points where the text is meaningful. The second column is separated at points where the participant decides to transfer the changes from his mind to the actual diagram.

If the tool does not offer any feature to apply the changes at once, he breaks down what he wants to do into smaller steps that his tool supports. Each cell of the third column shows the steps that the tool allows to be done at once. The fourth column shows the drawn diagram. The part of this process that takes place in the mind of the test participant does not depend on the tool, but when transferring from the mind to the actual diagram, the breaking down depends on the features of the tool. We call the operations that are independent of the tool *Model-space operation*, and the operations that depend on the tool *Tool-space operation*.

Each model-space operation maps to one or more tool-space operations. Depending on the tool that is being used and the skill of the user, the tool-space operations fulfilling one model-space operation may differ. For example, Figure 4.2 shows two possible ways of inserting a decision element between two already existing activities in a UML activity diagram. In one way, firstly, the connection between the two activities is removed, secondly, a new decision node is created, and finally, the activities and the decision element are connected together.

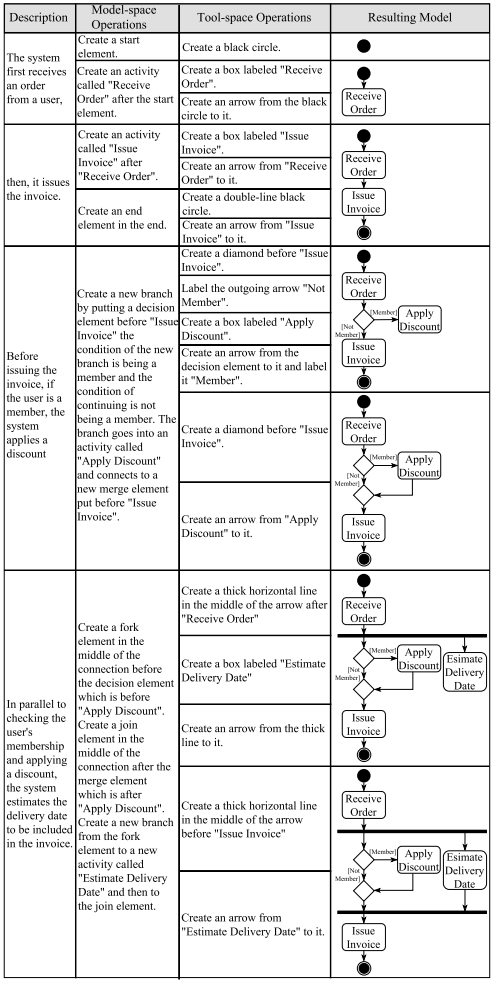


Figure 4.1: A sample scenario of drawing an activity diagram based on a natural language description of the underlying process

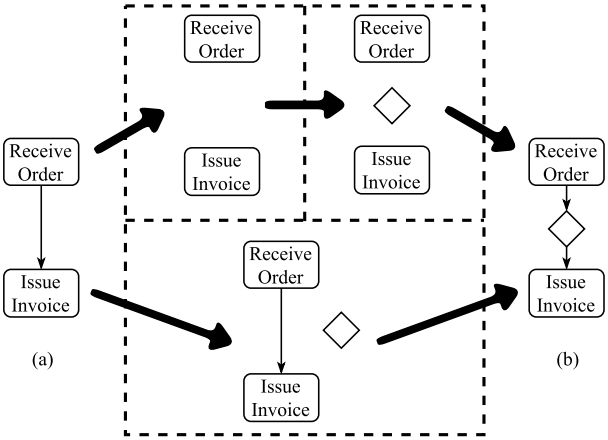


Figure 4.2: Inserting a decision node between two activities in two ways: (i) removing the connection, creating the decision node and connecting them together, and (ii) creating the decision node, dragging it over the connection and the tool automatically breaks the connection into two connections.

In the other way, the decision element is created first, and then, it is dragged over the connection between the two activities. The tool breaks the connection automatically and creates two new connections accordingly. In this example (Figure 4.2), when only considering diagram "a" that should be transformed into diagram "b", it is a model-space operation and does not depend on the tool being used. However, the intermediate steps are tool-space operations. The modeler can choose one of these two ways. We define the behavior of a graphical model as the mapping between model-space operations and the sets of tool-space operations that fulfill those model-space operations. In other words, we consider the behavior of two types of models to be similar when two equivalent

model-space operations are done by equivalent sets of tool-space operations. We continue by discussing two examples of how simple graphs behave differently from software engineering graphical models.

4.3.2 The Behavior of Simple Graphs

When a user interface designer wants to add a new feature such as the second insertion technique described in Figure 4.2 to his tool, he needs to be sure about the effectiveness of that feature before implementation. Therefore, he conducts tests with a prototype of the new technique. Figure 4.3 shows a part of a possible testing experiment when the tool designer uses a simpler equivalent graph instead of the activity diagram in order to implement the prototype quickly. The participant is given diagram "a" and is asked to insert another node between the existing nodes so that it transforms into diagram "b". While the testers expect the participants to do this task in one of the ways shown in Figure 4.2, they may do it as shown in Figure 4.3, which is not possible in the activity diagram example of Figure 4.2. The reason for this difference is that duplicating the node B, connecting the new node to the existing node B and renaming the existing node B to C is easier. Since the sets of tool-space operations that fulfill the model-space operations in the activity diagram and the simple graph are different, the behavior of these two models is not similar. Therefore, the result of this experiment is not valid for activity diagrams.

Figure 4.4 shows another example of simple graphs' behavior. In this example, the size of an element affects the behavior of the

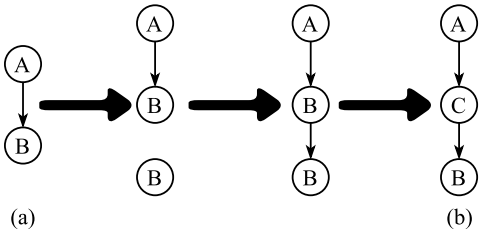


Figure 4.3: Inserting a node between two existing nodes in a simple graph can be done differently from the similar example of Figure 4.2

graphical model and consequently affects the user interaction. The task of this experiment is to add another activity after activity A3 so that diagram "a" transforms into diagram "c". Before adding the new activity element, the participants need to provide some space by moving the other elements. However, if the experiment is done on the equivalent graph "b" instead of an activity diagram, adding the node E after the node B can be done without moving other nodes resulting in "d". Due to the difference in behaviors of simple graphs and activity diagrams, conclusions from this experiment are not equally valid for activity diagrams.

By these two examples, we showed how simple graphs behave differently than the graphical models. Therefore, despite meeting other requirements, they cannot represent graphical models in manipulation usability tests. In the next section, we describe how we compensate for this shortcoming.

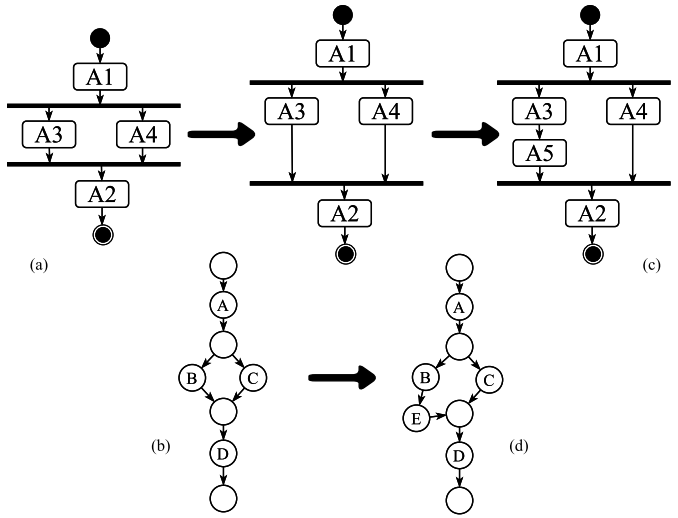


Figure 4.4: Similar tasks on an activity diagram and its equivalent graph may result in different layouts

4.4 Our Approach

Our proposal includes an extended definition of graph called *ImitGraph*, a way to specialize ImitGraphs for different purposes and a set of commands to define model-space operations. If a usability tester wants to use ImitGraphs defined in Section 4.4.1, he should first define his desired types of nodes and connections using the specialization method of Section 4.4.2, and then, design tasks for participants using the ImitGraph commands introduced in Section 4.4.3.

4.4.1 Definition of ImitGraphs

Inspired by the simplicity of the graphs, we extended the definition of graphs by adding more properties to the nodes and connections. The additional properties allow the usability testers to specialize ImitGraphs depending on the purpose of their tests and the graphical models involved.

ImitGraphs are composed of nodes, connections and joints. Figure 4.5 shows examples of each element.

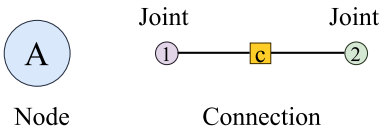


Figure 4.5: A node and a connection of a specialized graph

Node

A node is a circular element that can be assigned different sizes, colors, and it can hold a label.

Connection

A connection is a line with a rectangle in the middle. It connects two joints. The rectangle can be assigned different colors and a label.

Joint

A joint is a circle at the end of a connection. Joints attach connections to nodes. They can be assigned different colors and labels.

Since joints are used in defining connections and connections are used in defining the nodes, we first describe the properties of joints, then connections and finally the nodes.

Joints have the following properties: (i) Color: the color of a joint enables the user to distinguish between different types of joints. (ii) Label: if a graphical model requires joints to have textual parts (e.g., the cardinalities in a class diagram), their equivalent ImitGraph joints should hold labels instead.

Connections have the following properties: (i) Color: the color of the rectangle enables the user to distinguish between different types of connections. (ii) Label: if a graphical model requires the connections to have a textual part (e.g., the conditions in a flow chart diagram), their equivalent ImitGraph connections should hold a label. (iii) First joint: indicates the type of the joint at one end of a connection. (iv) Second joint: indicates the joint type for the other end of the connection. (v) Orientation: shows if a connection can be oriented in any direction or it is restricted to certain orientations (e.g., horizontal).

Nodes have the following properties: (i) Color: each node type has a different color so that the users can recognize their types.

(ii) Size: if the size of the nodes matter in an experiment they can be defined differently, otherwise, similar node sizes make experimenting simpler. (iii) Label: if the original graphical model's counterpart element holds a text, the ImitGraph node should hold a label. (iv) Connection type: each node is restricted to be connected with other nodes with certain types of connections. (v) Joint type: for each connection type, it indicates which joint of the connection should be connected to the node. (vi) Connection point: indicates if the connection can be connected to the node at any point on the perimeter or it is restricted to certain points (e.g., decision nodes in activity diagrams).

In graphical models, the group of elements that have a text can be referenced directly. The other elements are referenced relative to the nodes of the first group. This behavior is simulated by the ability of the nodes, joints, and connections to accept a label or not.

4.4.2 Specialization of ImitGraphs

Before usability testers use ImitGraphs in their experiments, different types of nodes, connections and joints should be defined by specifying the properties of each type. The properties are specified based on the elements and connections of the graphical models on which interaction techniques are going to be tested. Types should be defined adequately so that the behavior of the intended graphical models can be simulated. We called this phase *Specialization*. Figure 4.6 shows two examples of ImitGraphs and their original

graphical model: “a” is a part of an entity-relationship diagram (ERD) and “b” is a part of an activity diagram. Usability testers specify different colors of the elements and whether they accept labels or not.

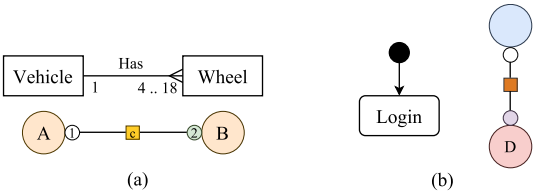


Figure 4.6: Parts of “a” an entity-relationship diagram and “b” an activity diagram, and their equivalent specialized graphs

The colors of the nodes in the ERD diagram’s equivalent ImitGraph are the same since the nodes are of the same type. In contrast, the colors of the nodes in the activity diagram’s equivalent ImitGraph are different. The colors of the joints in both ImitGraphs are different due to the different types of endpoints in the connections of the original models (e.g., flat, arrow and trifurcation). The nodes, joints, and connections have labels if their counterpart elements have texts.

The definitions of nodes and connections are reusable. Once the nodes and connections of a certain type of graphical model are defined, they can be used in other experiments that include the same type of graphical model.

4.4.3 Instructing Commands to Draw ImitGraphs

To make ImitGraphs applicable in usability tests we defined a set of commands for specifying the model-space operations. The usability testers use ImitGraph commands to instruct the participants to draw a graph. Although the graph that should be drawn in a testing experiment is equivalent to a software engineering graphical model, the participants are not aware of that. Since ImitGraphs do not have the semantics of their originals, the instructing methods of the original graphical models, e.g., the natural language description used in Figure 4.1, are not applicable. Therefore, we defined a set of ImitGraph commands to instruct the participants of tests. These commands do not suggest (i) a specific layout, (ii) specific tool-space operations, and (iii) a specific order of operations. The end user should have a similar freedom of choosing any tool-space operation, order and layout for drawing as he has when drawing based on a natural language description. They even should be able to make similar errors.

Before explaining the commands, we define two terms. *Current location* can be a node or a connection. It is the last node created, the node referenced by the last "Find Node" command, or the connection referenced by the last "Find Connection" command. *Referenceable location* is a type of node that can be located unambiguously by the participants. It can be a node with a label, current location, or a memorized location specified by "Remember as" command.

As the examples in Figure 4.7 show, the ImitGraph commands are partially textual and partially visual. Subsequently, we describe the commands in more detail:

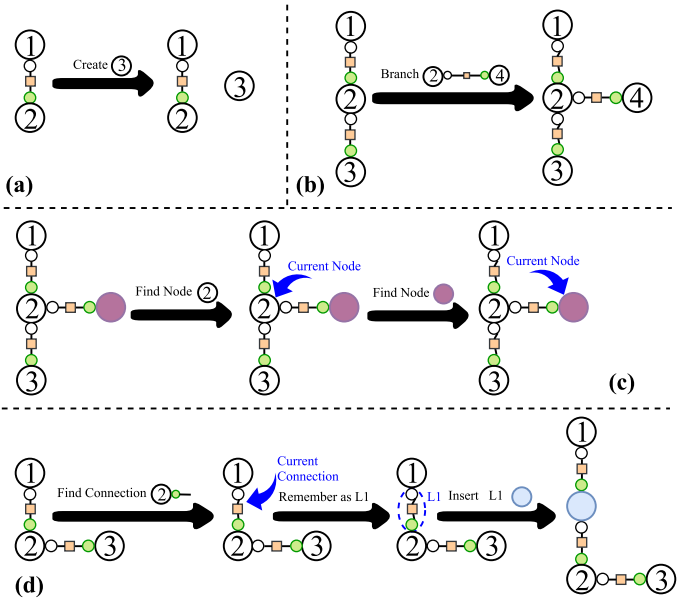


Figure 4.7: Examples of specialized graph commands that instruct participants how to manipulate diagrams: (a) Create a new node, (b) Branching from an existing node, (c) finding a node that is not referenceable, (d) finding a connection, assigning it a label and inserting a new node in the middle of it.

Recent

This command is followed by a node type that is not referenceable. It is used to reference a node that is created by the previous command and is basically not referenceable.

Create

As shown in Figure 4.7a, this command is followed by a node symbol with a color and a label if required. It instructs the participants to create a new node with the specified properties. It does not suggest any tool-space operation (e.g., it can be created by duplicating an existing node) or a position.

Branch

As shown in Figure 4.7b, this command is followed by a referenceable node. Then it continues with a sequence of connections and nodes. This command instructs the participants to add a branch made of nodes and connections to the diagram, which starts from a known node and ends in a known node or a new node. The participant is free to start from any of the connections or nodes of the sequence.

Find Node

Except for the nodes that are directly referenceable, other nodes should be referenced relatively. As shown in Figure 4.7c, this command is followed by a node. The node can be referenceable or not. If the specified node is referenceable, this command instructs the participants to consider it as the current location. Otherwise, the participant should find a node of the specified type that is connected to the current location and consider it as the current location.

Find Connection

As shown in Figure 4.7d, this command is followed by a referenceable node. Then the node is followed by a joint. This command instructs the participants to find the specified node, and then, find the connection that is connected with the specified joint to it. The found connection should be considered as the current location.

Remember as

As shown in Figure 4.7d, this command instructs the participants to assign the specified name to the current location in their mind. If the current location was not referenceable before, it can be referenced by this name afterward.

Insert

When this command is used, the current location should be a connection. As shown in Figure 4.7d, this command is followed by a new node and after that a sequence of connections, and nodes. It instructs the participants to add a new branch to the diagram. The beginning of the branch is a new node that should be placed between the nodes connected by the connection known as the current location at the moment. The end of the branch can be a new node or a referenceable existing node. This command does not suggest any order, layout or tool-space operation for creating the nodes and connections of the new branch.

Other commands

Remove instructs the participants to delete the current location. If the current location is pointing to a node, its connections should be deleted too. *Connect* followed by a referenceable node, a connection and another referenceable node, instructs the participant to create a connection between two existing nodes. *Replace* followed by a node or a connection, instructs the participants to replace the current location with the specified node or connection.

4.5 A Sample Usage Scenario

In this section, we demonstrate how usability testers can use ImitGraphs in usability testing experiments. For this purpose, we

made a hypothetical experiment, in which the usability testers intend to test the usability of their tool's interaction techniques when manipulating activity diagrams. For this purpose, they ask the participants to draw an ImitGraph. Before the tests, the testers specialize the ImitGraph to imitate the behavior of activity diagrams by defining equivalent joints, connections, and nodes. Then, they use ImitGraph commands to create a task based on the process of Figure 4.1.

The participants are not aware of the relationship between the graph that they draw and the activity diagram. They first study the definition of the nodes, connections and joints. Then, they read the commands one by one like a natural language description, and at certain points decide to perform the commands that they have read. Since the commands only instruct the participants to imagine an addition or change, the decision on how to perform them is made based on the available tool features. The first column of Figure 4.8 shows the task made of commands. The second column shows how participants can split the task into model-space operations. The third column shows tool space operations that participants perform to fulfill each model-space operation. In this example, we assumed that the tool under test is a simple drawing tool that can create nodes, connect them and insert a node in the middle of a connection. Different participants can split the task in different ways and perform the model-space operations with different tool-space operations. Figure 4.8 only shows one possible way.

The specialized ImitGraph defined by the usability testers based on our activity diagram example (Figure 4.1) includes two types of

joints which are presented in Table 4.1, two types of connections which are presented in Table 4.2, and six types of nodes which are presented in Table 4.3.

Table 4.1: Joint Types

Type	Symbol	Label
J1	○	no
J2	●	no

Table 4.2: Connection Types

Type	Symbol	Label	First joint	Second joint	Orientation
C1	□	optional	J1	J2	any
C2	■	no	J1	J1	horizontal

In Table 4.1, the white joint type (J1) represents the simple flat end of the connections in activity diagrams. The green joint type (J2) represents the arrow end of the connections. None of the joints can have a label in this example.

In Table 4.2, the beige connection type (C1) represents the connections of activity diagrams which have a flat end (J1) and an arrow end (J2). They can be oriented in any angle and can hold a label when they are connected to the equivalents of decision nodes. The red connection type (C2) is used to represent fork/join elements of activity diagrams. They are flat at both ends (J1) and only can be oriented horizontally.

In Table 4.3, the gray node type (N1) represents the start element of activity diagrams which does not accept a label. It can have only one simple arrow connection (C1), which connects to the node by its flat end (J1) and at any point of the node's perimeter.

The purple node type (N2) represents the end element of activity diagrams. It is similar to N1 but connects to the arrow from the arrow end (J2).

Table 4.3: Node Types

Type	Symbol	Size (pixel)	Label	Connection type	Joint type	Min	Max	Connection point
N1		30	no	C1	J1	1	1	any
N2		30	no	C1	J2	1	1	any
N3		30	yes	C1	J2	1	1	any
				C1	J1	1	1	any
N4		30	no	C1	J2	0	1	top
				C1	J1	0	1	bottom
				C2	-	1	2	left, right
N5		30	no	C1	J2	1	1	top, left, right, bottom
				C1	J1	2	3	top, left, right, bottom
N6		30	no	C1	J2	2	3	top, left, right, bottom
				C1	J1	1	1	top, left, right, bottom

The white node type (N3) represents the activity elements of activity diagrams. They hold labels and have one incoming connection and one outgoing connection of type C1. Incoming connections connect with a J2 joint and outgoing connections connect with a J1 joint. The connections can connect at any point to this type of nodes. The blue node type (N4) are used in creating equivalents of fork/join elements of activity diagrams. Multiple nodes of this type represent one fork/join element. For this purpose, they are connected together with C2 connections that can only be oriented horizontally and are connected only from left and right.

The number of N4 nodes that represent a fork/join element depends on the number of incoming and outgoing connections of the fork/join element. The incoming connections of type C1 connect with a J2 joint to the top. The outgoing connections of type C1 connect with a J1 joint to the bottom. The orange (N5) and the yellow (N6) node types represent decision and merge elements of activity diagrams. Therefore, they are restricted to be connected at the top, left, right and bottom only.

This example shows that specialized ImitGraphs can be drawn in the same way as their original graphical model. In addition, ImitGraph commands can instruct the participants to draw a graph without suggesting any layout, operation or order. The participants are responsible for choosing appropriate tool-space operations. The definition of the joints, connections, and nodes make the participants draw the diagrams with a layout similar to the original graphical model.

4.6 Conclusions and Future Work

User interface researchers and tool developers who work on improving the usability of software modeling tools need to conduct tests to evaluate the effectiveness of their new ideas and gather feedback to improve them. In this paper, we proposed ImitGraphs, an extended version of graphs that can be used in such tests instead of real graphical models. The benefits of using this approach are: (i) fast development of a working prototype to evaluate new ideas

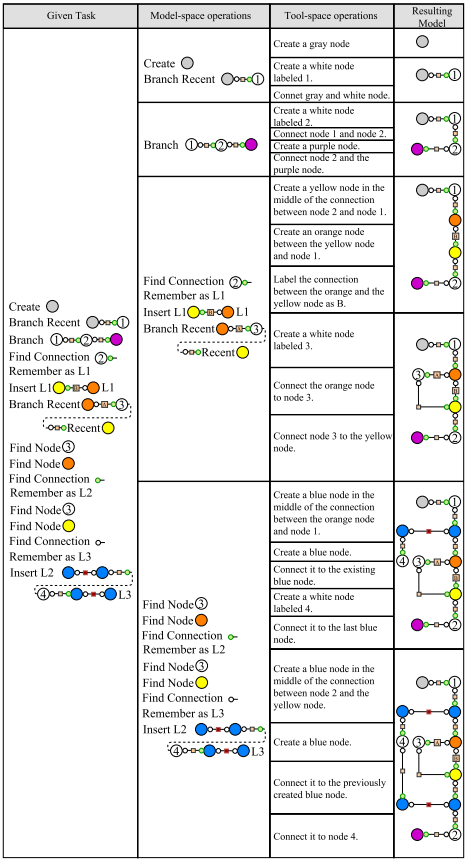


Figure 4.8: Equivalent scenario of Figure 4.1 in ImitGraph notation. The first column contains the task given to the participants. The second column shows the detected model-space operations by the participant. The third column shows the corresponding tool-space operations. The last column shows the resulting diagram at each step.

resulting in a lower cost and earlier feedback, (ii) evaluating the effectiveness of the new ideas on a wider range of graphical models, and (iii) possibility to recruit participants with no prior knowledge about the intended graphical models for the experiments. In exchange for these benefits, tool developers (a) have to teach ImitGraphs definition and its commands to the participants before their first experiment, and (b) must develop a working prototype for ImitGraphs which is not a part of the final product. The former can be compensated by enabling testers to recruit participants more easily since more people fit. The latter can be justified by the low cost of developing a tool for ImitGraphs due to their simple appearance and the fact that the developed prototypes and the definitions of the ImitGraphs will not be thrown away. They will be used in further optimizations and tests of future improvements of the user interface. Even other user interface designers and researchers can benefit from the already developed prototypes and definitions to rapidly test their ideas at a very early stage.

Since ImitGraphs imitate the interaction behavior and layout properties of graphical models, the gained experience and feedback can be transferred to the original models. Therefore, ImitGraphs can be used to test a wide range of user interface interaction features such as drawing commands, automatic alignment of elements, placement of connections, readjustment of the layout when changes occur or when more space is required and providing frequently done operations as a single command. The presentation of the tools such as placement of the buttons and activators, the structure of the menus, the expressiveness of the icons, the way of showing hints and extra information can be tested, in addition to their features.

Furthermore, complex features such as graphical search and filter can be optimized by conducting tests using ImitGraphs.

This study will continue with specializing ImitGraphs for some of the most frequently used graphical models in software engineering, developing tool-support for manipulating ImitGraphs, and finally, use them in real experiments to evaluate ImitGraphs.

Chapter 5

Common User-Interface Features of Requirements Modeling Tools

Original publication:
Choosing Requirements for Experimentation with User Interfaces of Requirements Modeling Tools
P. Ghazi, Z. Shakeri Hossein Abad and M. Glinz
International Requirements Engineering Conference 2017

Abstract

When designing a new presentation front-end called FlexiView for requirements modeling tools, we encountered a general problem: designing such an interface requires a lot of experimentation which is costly when the code of the tool needs to be adapted for every experiment. On the other hand, when using simplified user interface

(UI) tools, the results are difficult to generalize. To improve this situation, we are developing an UI experimentation tool which is based on so-called ImitGraphs. ImitGraphs can act as a simple, but accurate substitute for a modeling tool. In this paper, we define requirements for such an UI experimentation tool based on an analysis of the features of existing requirements modeling tools.

5.1 Introduction

Requirements engineers spend a lot of their time working with modeling tools. Thus, the usability of their modeling tools affects their productivity [dGNA+12]. However, the User Interface (UI) of this type of tools has not changed for a long time despite the challenges that exist in working with artifacts [GG16]. Information presentation is one of the aspects of the modeling tools that can be improved. We are developing a new tool front-end called FlexiView [GSG15] for using the screen space efficiently by presenting information in heterogeneous levels of detail. Like every other new feature, it should go through multiple cycles of usability experimentation and optimization in order to mature.

The high cost of usability experiments at the early stages of software development is one of the reasons that the improvement of the UIs of modeling tools are neglected. We have proposed ImitGraphs [GG17b] to lower the cost of usability experiments. ImitGraphs are an extended version of Graphs that can substitute Requirements Engineering (RE) graphical models (e.g., diagrams such as activity diagrams and sequence diagrams) in usability experiments. The simplicity of ImitGraphs enables usability testers to quickly develop experimental tools instead of using the modeling tools as a testing platform. We intend to design an experimental tool based on ImitGraphs for testing and optimizing FlexiView. Since FlexiView will be integrated into modeling tools, the experimental tool that we design should have features similar to the features of existing modeling tools. Our goal in this paper is to

study the basic features of the existing modeling tools and define the requirements of a suitable experimental tool by including the most frequent features.

To achieve this goal, we conducted a market study in which we analyzed the UI features of a group of modeling tools. Then, we selected the features with the highest frequency as the UI requirements of the tool that we need for experimenting on Flexiview. Our contributions are (i) a list of basic essential manipulation actions in modeling tools, (ii) different methods of performing those actions with their frequencies, and (iii) the UI requirements of an experimental tool for testing UI features.

5.2 Approach

We performed our study of basic features of existing modeling tools in three steps: selecting tools, defining basic manipulation actions and finding the most frequent method for each action. In **step 1**, we defined the criteria of selecting tools for our study. Since FlexiView is designed to be used in touch screen modeling tools, we searched Google Play Store with the keywords "UML" and "diagram" and picked apps with an average score of 3.6 and above. We ended up with the following list of ten modeling tools: Flowdia Lite (T1), Draw Express Lite (T2), FlexiSketch (T3), Droiddia (T4), Grapholite (T5), Draw.io (T6), Lekh Diagram (T7), Diagrid (T8), ClickCharts Free (T9), and NodeScape (T10). In **step 2**, we defined a set of basic manipulation actions that are

essential for a modeling tool by watching modeling tutorials on YouTube. The videos were not related to the tools under study and were intended for beginners. We extracted the following list of eleven basic essential actions from the videos: creating a new object, opening a context menu, scrolling, deleting an existing object or an existing connection, selecting multiple objects, duplicating an object, changing the color of an object, changing the style of a connection, moving an object, connecting objects, and adding/changing the text of an object or a connection. In **step 3**, we inspected the tools and identified the methods by which the actions can be performed.

5.3 Results

The result of our observations is presented in Table 5.1. The columns contain essential actions, the methods of performing those actions, the number of the tools that employ those methods, and the corresponding tools. For example, creating an object can be done differently, e.g., by dragging the object from the menu and dropping it on the canvas, by selecting the object from the menu, by freehand drawing the object, by long touching a location on the canvas and selecting the object from the pop-up menu, or by selecting the object from the menu and then touching a location on the canvas.

Some actions can be performed in more than one way in some tools. This is the reason why, for some actions, the sum of the frequencies

Table 5.1: Basic actions and how they can be performed in tools

Action	Method	#	Tool
Create an object	Drag and drop from the menu	5	T1-2, T6, T9-10
	Select from the menu	5	T1-2, T5-7
	Free draw	2	T3, T7
	Long touch then select from the pop-up menu	1	T4
	Select from the menu then single touch	3	T3, T8-9
Open the Context menu	Same time as selecting the object	8	T1-5, T7-8, T10
	Single touch on the selected object	1	T6
	Not available	1	T9
Scrolling	One finger	7	T1, T4-10
	Two fingers	3	T2-3, T6
Delete object/connection	Select from the context menu	8	T1, T3-8, T10
	Select from the menu	2	T2, T9
Select multiple objects: *Step 1-initiate selection*	Long touch on the canvas	3	T5-6, T9
	Select from the menu	5	T1-2, T7, T9-10
	Not available	3	T3-4, T8
Select multiple objects: *Step 2-indicate objects*	Free hand	1	T2
	Rectangle	6	T1, T5-7, T9-10
Duplicate an object	Select from the context menu	6	T1, T4-7, T10
	Gesture command	1	T2
	Not available	3	T3, T8-9
Edit object color	Select from the context menu	5	T1, T4, T7-8, T10
	Edit directly in the side menu	4	T2, T5-6, T9
	Double click on the object	1	T8
	Not available	1	T3
Change a connection's style	Select from the context menu	6	T1,T3-4,T7-8,T10
	Edit directly in the side menu	4	T2, T5-6, T9
	Double click on the connection	1	T8
Move an object: *Step 1-initiate move*	Move selected	7	T2, T3, T5-8, T10
	Move unselected	3	T1, T6, T9
	Long touch	1	T4
Move an object: *Step 2-move*	Dragging the handle	2	T2, T10
	Dragging the object	8	T1, T3-9
Connect two objects	Drag the handle	2	T5-6
	Select from the menu	1	T6
	Select from the menu then draw a free hand conn	2	T9-10
	Select from context menu then select second obj	3	T1, T4, T8
	Draw free hand connection	3	T2-3, T7
Change the text	Double click on the object and connection	7	T2, T4-6, T8-10
	Select from the context menu	4	T1, T3, T7, T8

is more than the number of the studied tools. Some actions can be performed in separate independent steps. For example, for selecting multiple objects, first, the user initiates the selection, then, indicates the objects. The first step can be done by a long touch, or by selecting the corresponding icon from the menu. The

second step can be done by drawing a freehand lasso around the objects, or by drawing a rectangle around the objects. Considering such steps separately allowed us to find the most frequent methods more accurately than by analyzing actions only, since an action might be rather infrequent while one of its constituent steps occurs frequently.

5.4 Choosing the Requirements

Based on the frequency of the methods, we chose the requirements for our experimental tool. During this process, we encountered two special cases. First, when two methods conflicted, and second when there was a tie. In the case of a conflict, we chose the combination of methods with a higher overall frequency. For example, we cannot have scrolling with one finger and connecting objects by freehand drawing at the same time. We had to choose between (i) one-finger scrolling and using the context menu for connecting objects, or (ii) two-finger scrolling and connecting objects by freehand drawing. In this case, we chose the first combination based on the higher overall frequency.

In case of a tie, if implementing both of the options was possible, we chose both. For example, creating an object by drag-and-drop or by selection from the menu could co-exist in a tool. Therefore, we chose both of them. If methods with equal frequencies could not co-exist in a tool, we used the score of the tools to break the tie. Finally, our study resulted in the following requirements for

our experimental tool.
1. The tool should allow users to create an object by drag-and-dropping it from the menu onto the canvas and also by selecting the icon of an object from the menu. 2. A context menu should appear when an object is selected. 3. The user should be able to scroll using one finger. 4. An object can be deleted by selecting the corresponding command from the context menu. 5. To select multiple objects, the user should first select the corresponding command from the menu and then draw a rectangle around the desired objects. 6. In order to duplicate an object, the corresponding command should be selected from the context menu. 7. The user should be able to change the color of the objects after selecting the corresponding command from the context menu. 8. The user should be able to change the color and type of the connections after selecting the corresponding command from the context menu. 9. The user should be able to move a selected object by dragging. 10. In order to connect two objects, first the corresponding command should be selected from the context menu and then the second object should be selected. 11. The user should be able to change the text of the objects and connections after double-tapping on them.

5.5 Conclusion and Future Work

When a feature such as FlexiView will be eventually integrated into other modeling tools, the generalizability of the usability experiments is important. Therefore, the UI features of the experimental tool should be as similar as possible to the features

of the target modeling tools. In order to design such a tool, we studied available modeling tools and extracted the most frequent methods of performing essential actions in those tools. Based on the results, we defined the UI requirements of an ImitGraphs-based, experimental tool that can be used for experimentation with the UI of RE tools.

This work will be continued by actually implementing a tool based on the defined requirements and conducting usability experiments with the UI of FlexiView.

Chapter 6
FlexiView Experimental Tool

Original publication:
FlexiView Experimental Tool: Fair and Detailed Usability Tests for Requirements Modeling Tools
P. Ghazi and M. Glinz
International Requirements Engineering Conference 2018

Abstract

Enhancing the usability of tools such as requirements modeling tools requires several cycles of testing and improvement. Since this process is costly, it is usually ignored.

In this paper, we present an experimental tool which we have developed with two goals: (i) comparing the usability of a new navigation technique for requirements artifacts called FlexiView with traditional zooming and scrolling, and (ii) developing a platform that enables fast implementation and fair usability comparisons of new navigation techniques while producing generalizable results.

6.1 Introduction

Requirements engineers elicit various types of information, analyze them and store them in artifacts. In order to conveniently read, understand, modify and share the information in these artifacts, they choose suitable formats such as documents, charts, and diagrams [dGNA+12], and use requirements modeling tools (RMTs) for working with these artifacts. Thus, RMTs greatly influence the performance of requirements engineers.

In addition to the suitability of the functional features of a tool, its usability also impacts the performance of doing a task. In the definition of usability by the ISO 9241-11 standard [Int18], three properties of using a product are considered: efficiency, effectiveness, and satisfaction. To improve the usability of a tool, several cycles of testing and improvement are required. In each cycle, (i) new solutions for the known challenges are implemented, (ii) usability tests are carried out and (iii) the solutions are updated based on the results of the tests.

Different aspects of an RMT influence the usability factors. We focus on the following aspects: (i) how the information is presented on the screen and (ii) how the users can customize this presentation to fit their needs at a certain moment. The process of usability testing and improvement is costly and thus mainly ignored when developing RMTs. As a result, the graphical user interfaces of RMTs have not changed significantly since their emergence in the 1980s, although users experience major challenges especially

when working with interconnected artifacts that are larger than the screen [GG17a].

In this paper, we present the tool that we developed for performing usability tests on FlexiView [GSG15]. FlexiView is a new physics-based focus+context navigation technique for large and interconnected requirements artifacts. In FlexiView, based on the current task, the user indicates interesting elements of a graphical model by putting virtual magnets on them. Based on the strength of the magnets, FlexiView allocates screen space to the elements. The result is a view of the diagram with all elements, in which the more interesting elements are larger and reveal more details than the other elements.

Our goal was twofold: (i) creating a platform for evaluating FlexiView by comparing it to traditional navigation techniques and (ii) creating a framework for enhancing the general process of usability testing and improvement cycles for RMTs by making usability testing cheaper and faster. In order to decrease the cost of usability test cycles, a framework is needed that allows *fast* implementation and modification of solutions. Meanwhile, it should provide *fair* comparisons with *generalizable* results.

For having fairness in comparisons, the solutions that are being compared should be tested in equal situations. Implementing the solutions in different tools with different features does not guarantee that the observed differences are only due to the differences between the solutions. Therefore, both solutions should be either implemented in an existing RMT or in an experimental tool. In

the first case, since an existing tool is already complicated, integrating a new solution into it and modifying it after a round of usability tests is time-consuming. In the second case, while implementing solutions is faster in an experimental tool, the test cases are limited due to the lack of advanced features, which reduces generalizability.

We decided to implement zooming, scrolling, and FlexiView navigation in an experimental tool for having *fast* implementations and *fair* comparisons. We used ImitGraphs instead of requirements models to preserve the *generalizability* of the results. ImitGraphs are versions of simple node-and-edge graphs with additional parameters that allow them to imitate the behavior of other models [GG17b].

6.2 FlexiView Tool

In essence, our tool is an ImitGraphs modeling tool for medium-sized tablets with zooming, scrolling, and FlexiView navigation techniques. It can log user interactions with high detail and allows the usability tester to define tasks. We call it *experimental* as it permits easy and fast experimentation.

Our tool has two types of users: usability testers and users of the usability tests. The usability testers have three major use cases: (i) customizing ImitGraphs by defining different types of nodes, connections and joints, (ii) creating different usability testing tasks

by defining the input files, the time limit and which navigation techniques are allowed in each task, (iii) loading tasks and starting them during the usability tests.

The users of the usability tests start after the usability tester loads a task. Figure 6.1 shows three screenshots of the tool with highlighted numbers (1-10). The left-side menu contains the nodes (1) and connections (2) that are defined by the testers. In the middle of the screen, the editor area is located (3). The users can create new nodes and connections by dragging them from the side menu to the editor area. Within the editor area, the users can move nodes, joints, and connections by one-finger dragging. If a node and a joint of a connection are allowed to be attached in the ImitGraph definition, the user can attach them by moving the joint close to the node. When tapping on nodes and connections, a context menu appears above the selected object (4). The users can remove an object and edit its text by pushing the corresponding buttons (5 and 7 respectively) in the context menu.

In Figure 6.1a, the editor area contains an ImitGraph composed of four nodes and four connections. In Figure 6.1b, the user zoomed in on the upper-left node in order to see its details. Zooming and scrolling are possible by a pinch gesture.

In the FlexiView navigation, three parts of the user interface are involved: context menu, magnet pane (8) and magnet slider (9). In Figure 6.1c the user magnetizes the upper-left node to see its details. To achieve this, first, a magnet is put on a node by pushing the corresponding button in the context menu (6). Then, via the slider

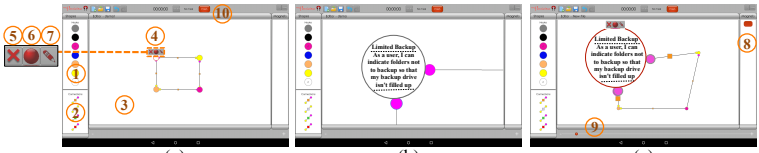


Figure 6.1: Three screenshots of the tool while showing a simple diagram (a) and while the upper-left node is zoomed in (b) and magnetized (c).

at the bottom (9), the strength of the magnet can be adjusted. As soon as a magnet is created, a red button appears in the magnet pane (8) which represents that magnet. When a user clicks a magnet button in the magnet pane, the node in the model which carries the corresponding magnet is selected. Thus, the magnet pane provides an overview of all currently set magnets.

During the usability test trials, the tool automatically collects experiment data by recording every interaction of the users with millisecond precision. The recordings comprise two types of interactions: instant and continuous. Instant interactions do not have a duration, e.g., selecting a node. The continuous interactions take place over a period of time which can be less than a second or up to several seconds, e.g., moving a node. The tool records one entry for each instant interaction and multiple entries for continuous interactions depending on their duration. The task menu at the top (10) shows a timer and a finish button. The timer encourages the users to perform the task as fast as they can.

Different types of information can be extracted from the raw data recorded by our tool. Let us consider this scenario: *The user is*

given a diagram with a hierarchical structure and is asked to search the hierarchy to find the nodes with a certain keyword using zooming and scrolling. In this scenario, the user has to zoom in to check the details of each node. When zoomed in, the connections of the nodes are not recognizable. Therefore, after checking a node, the user needs to zoom out to view the overall structure of the diagram and plan for visiting the next nodes. The user continuously zooms in and out until all nodes are checked. From the recorded data we can find out how much time the user spent on checking, planning and navigating. Additionally, we can visualize the users' search path and count the nodes that they missed or checked more than once. A similar scenario can be devised for FlexiView which creates the opportunity of comparing these techniques.

6.3 Conclusion and Future Work

We have conducted first experiments with the FlexiView tool and found that it is capable of gathering valuable user interaction data with high accuracy and detail [GG18]. This proved to be helpful in comparing the usability of FlexiView in comparison with classic zooming and scrolling. Using ImitGraphs speeded up the process of creating our experimental tool while it allowed us to experiment with ImitGraphs that imitated the behavior of requirements engineering models such as activity diagrams, class diagrams, and goal decomposition models. Finding the exact extent to which the results are generalizable needs further research. Next, we want to use our tool for conducting comparisons of other navigation techniques.

Chapter 7

An Experimental Comparison of Two Navigation Techniques for Requirements Modeling Tools

Original publication:
An Experimental Comparison of Two Navigation Techniques for Requirements Modeling Tools
P. Ghazi and M. Glinz
International Requirements Engineering Conference 2018

Abstract

In Requirements Engineering, many modeling tasks require viewing different parts of a model concurrently. However, traditional zoom+scroll navigation uses a single focus zoom, i.e., at a given

point in time, a user can zoom in on a single spot in the model only. Therefore, new focus+context navigation techniques have been proposed that allow multiple foci at the same time.

In this paper, we report on an experiment with students where we compare the participants' performance when using a requirements modeling tool with traditional zoom+scroll navigation vs. one with so-called FlexiView navigation which is a focus+context technique with multiple foci. The participants had to perform typical modeling tasks such as searching, editing, and traversing a model. All tasks were performed on medium-sized tablets with a tool for manipulating so-called ImitGraphs. ImitGraphs are enriched node-and-edge diagrams that can mimic various diagram types such as class, activity, or goal decomposition diagrams. We found that navigation with FlexiView outperformed zoom+scroll navigation with respect to task completion time, number of mistakes, cognitive load, and user satisfaction.

7.1 Introduction

In the process of Requirements Engineering (RE), requirements engineers create various types of requirements artifacts, e.g., textual specifications, glossaries, charts, and models [Lis15] [GG16]. Despite the popularity of natural language for expressing requirements, graphical requirements models play a significant role in RE and requirements engineers spend a noticeable amount of time working with modeling tools [dGNA+12]. In addition to the functional features of a tool, its user interface (UI) features influence the usability factors such as efficiency, effectiveness, and satisfaction [FHH00] [Int18]. A particular UI problem is how to deal with models that are larger than the available screen. This frequently happens even with large display screens and means that navigation mechanisms such as scrolling and zooming are needed. When working with small screen devices such as tablets, navigation in the model becomes a major challenge [GG17a].

Navigation with classic zooming and scrolling has been criticized for losing context when zooming in and its inability to view more than one part of the model in detail concurrently. To address this problem, so-called focus+context navigation techniques have been proposed [Fur86] [CKB09] [RMS+08] [GSG15]. These techniques provide zooming with multiple foci and preserve the context of the zoomed-in regions by displaying it with less detail. However, the effects of using novel navigation techniques on the performance of users when carrying out typical requirements modeling tasks have not been studied empirically so far.

In this paper, we report on a study where we compared the performance of two navigation techniques in an experiment with students. For our comparison, we chose a novel focus+context technique designed for requirements artifacts called FlexiView [GSG15] and a traditional navigation technique composed of zooming and scrolling. FlexiView provides multiple foci zooming based on a physical metaphor of magnets and springs. It uses a canvas of fixed size, so there is no need for scrolling.

Model navigation is a problem that occurs in any graphical modeling language, not just in RE. We performed our experiment in an RE context due to the importance of modeling in RE and because we eventually wanted to know whether requirements engineers can perform modeling tasks faster and with fewer errors when using a novel navigation technique such as FlexiView than when working with a traditional one.

We found in our experiment that FlexiView significantly outperformed traditional navigation with zooming and scrolling with respect to task completion time, number of errors made and overall satisfaction of users. We also found that using FlexiView reduced the cognitive load of the experiment participants when performing memory-intensive tasks and that they considered FlexiView to be easy to learn.

The remainder of this paper is organized as follows. We review navigation techniques in Sect. 7.2. Sect. 7.3 describes the goals of our study and our research questions. In sections 7.4 and 7.5, we present the language, tool, and modeling tasks used, while

Sect. 7.6 describes the experiment. In Sect. 7.7 we present and discuss our results. Sect. 7.8 deals with the threats to validity and Sect. 7.9 concludes.

7.2 UI navigation techniques

Graphical requirements models are typically larger than the available screen [GG17a]. The user interfaces (UIs) of tools for editing and viewing such models, therefore, need to provide mechanisms for navigating in large models. Also, the UIs of requirements modeling tools should provide a mechanism for viewing different parts of a model in detail concurrently, as requirements engineers frequently need this when editing or analyzing a model. Cockburn et al. [CKB09] reviewed different navigation techniques and categorized them. In this section, we briefly characterize typical navigation mechanisms for graphical requirements models.

7.2.1 Explosive Zooming with Multiple Windows

The early graphical requirements modeling tools (e.g., Software Through Pictures or Teamwork) used multiple windows for navigating in hierarchically structured models. For example, when users wanted to view the details of an activity in a dataflow diagram, they clicked that activity and the tool displayed the underlying dataflow diagram or textual spec in a new window. Scrolling was used when a diagram was larger than the window displaying it.

This so-called explosive zooming is easy to implement and use. It also provides a straightforward way of viewing details of different parts of a model concurrently by arranging the respective windows side by side. The big disadvantage is that users are quickly lost in a pile of overlapping windows. Also, on mobile devices with small screens, multiple windows are not equally usable.

7.2.2 Zooming and Scrolling

Navigating in a graphical structure by proportionally enlarging (zooming-in) or shrinking (zooming-out) the whole structure around a focus point, is a proven navigation technique which has its origin in graphics drawing tools. The focus point can be the center of the screen or a selected model element. Proportional zooming-in enlarges the drawing canvas, which then may become larger than the available display device. Hence proportional zooming must be combined with a scrolling mechanism. In the remainder of this paper, we speak of *ZoomScroll* when referring to the combination of proportional zooming and scrolling. ZoomScroll is today's standard technique for navigating in graphics, images, maps, etc. on smartphones and tablets. Using two-finger gestures, it is very easy and straightforward to use. However, ZoomScroll is a single focus technique, i.e., it does not allow to zoom in on more than one part of the model concurrently.

7.2.3 Focus+Context

To overcome the shortcomings of explosive zooming (loss of context) and proportional zooming (single focus only), so-called *focus+context* techniques have been developed for navigation in models [CKB09]. The goal is to provide zooming-in on details with multiple foci, while preserving the full context around the model elements which are currently zoomed-in. The idea is to visualize the focused parts of the model with high detail while the surrounding context is visualized without any details. Furnas [Fur86] developed fisheye views, the first focus+context visualization technique, in 1986. In Requirements Engineering, Onion graphs [KM07] have been proposed as a focus+context technique for visualizing large UML models. This technique visualizes the focus area with UML notation and the rest with their own notation. In ADORA [RMS+08], zooming-in works by enlarging the focused object (thus creating space for displaying information contained in that object) and pushing away all other objects that would overlap with the enlarged object otherwise. However, with every zooming-in operation, the model canvas becomes larger, so this technique also must be combined with scrolling and does not work well for small display screens. ADORA supports zooming with multiple foci. FlexiView [GSG15] is a focus+context technique for visualizing a set of connected artifacts. FlexiView is based on a physical metaphor of magnets and springs [RK14] [FR91], using a canvas of fixed size. When a user zooms in on some region of the canvas, this is interpreted as applying a magnetic force at this point which pushes the surrounding regions towards the borders

of the canvas and, as the canvas size is fixed, shrinks them. Thus FlexiView does not need scrolling. The disadvantage is that zooming in FlexiView geometrically distorts the model. The advantage is that an overview of the artifact always exists on the screen in which the relative positions of the components are preserved. FlexiView also supports zooming with multiple foci. Figure 7.1 shows how zooming in FlexiView works.

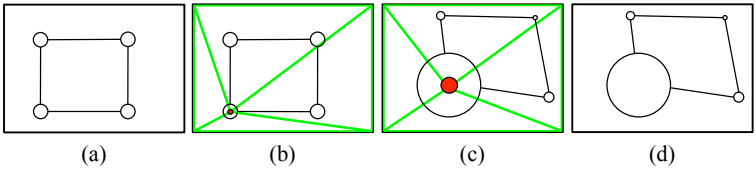


Figure 7.1: FlexiView's magnifying process: (a) A graph. (b) A magnet is put on the lower left node. The green lines (not visible to users) show how FlexiView partitions the space. (c) The power of the magnet is increased, causing the focused node to enlarge and other nodes to shrink. (d) How the magnified graph is actually displayed (without the green lines).

7.2.4 Semantic Zooming

When zooming-in not just enlarges the focused area but uses the enlarged space to display additional information, we call this *semantic zooming* [BH94]. In ADORA, for example, zooming-in into an object results in displaying sub-models contained in that object. In FlexiView, the subject of semantic zooming can be a whole artifact or a component of an artifact. In the case of a whole artifact, semantic zooming reveals the constituents of the artifact

and in the case of a component, semantic zooming reveals more details of the component.

7.2.5 Comparing navigation techniques

There is a limited number of studies on navigation techniques of requirements modeling tools such as the mentioned ADORA navigation technique [RMS+08], Onion graphs for UML diagrams [KM07] and hierarchical visualization of artifacts in automatic tracing tools [CHH07]. To our best knowledge, there exists no empirical comparison of the effects of different navigation techniques on the performance of carrying out typical modeling tasks. Such a comparison is inherently difficult because navigation techniques typically come with specific tools, supporting specific modeling languages. So when comparing them, it is almost impossible to determine whether the observed effects are caused by the navigation techniques or by other differences in the tools and languages.

For our study, we decided to compare a classic navigation technique such as ZoomScroll with a focus+context navigation technique. For the latter, we chose FlexiView, as FlexiView does not use scrolling and works well on tablet devices which we wanted to use. To avoid the comparison problems mentioned above, we used a generic graphical modeling language and created a tool for editing and viewing models that implements both ZoomScroll and FlexiView navigation.

7.3 Study Goals

The main goal of this study is to compare two navigation techniques for graphical requirements models with respect to the performance of users when carrying out typical modeling tasks. We particularly study whether a focus+context navigation technique such as FlexiView outperforms a classic navigation technique such as ZoomScroll. We compare these two navigation techniques in terms of efficiency and effectiveness. In addition, we want to get insight into how users rely on their memory for search path planning and working with parts of artifacts that cannot be accommodated in a single view on the screen. We framed our study goal in three research questions:

RQ1 *Can a focus+context navigation technique improve the efficiency and effectiveness of working with requirements artifacts compared to zooming and scrolling?*

RQ2 *Can a focus+context navigation technique decrease the cognitive load of working with requirements artifacts compared to zooming and scrolling?*

RQ3 *Will requirements engineers be more satisfied with using a focus+context navigation technique compared to zooming and scrolling?*

With RQ1 we aim at collecting quantitative empirical evidence, testing the hypothesis that focus+context navigation with FlexiView outperforms classic ZoomScroll navigation. In RQ2, we study how much a focus+context navigation technique influences the way

users rely on their memory in typical RE tasks. RQ3 investigates the users' attitude towards using a navigation technique which is different from the ZoomScroll navigation that they use in their daily digital lives.

For answering our research questions, we designed and conducted an experiment with 24 students. A crucial prerequisite for such an experiment is the availability of a tool that supports a representative modeling language and provides both FlexiView and ZoomScroll navigation. The language and tool used in our experiment are described in the next section. Another prerequisite is the definition of modeling tasks to be carried out in the experiment. We present the four tasks we designed in Section 7.5. Then, we present the details about our experiment in Section 7.6.

7.4 Language and Tool

In our study, we used ImitGraphs [GG17b], a generic modeling language that can mimic widely used modeling languages such as UML class or activity diagrams. Their simple appearance makes the development of experimental tools faster and their ability to imitate multiple graphical models ensures generalizable results. Using ImitGraphs drastically simplified the development of the tool required for our experiment and at the same time allowed us to carry out typical requirements modeling tasks on a model representing three types of models.

ImitGraphs are composed of three types of components: nodes, connections, and joints. Figure 7.2a shows a sample node. Figure 7.2b shows a sample connection which is composed of two joints (one at each end) and a central rectangular piece.

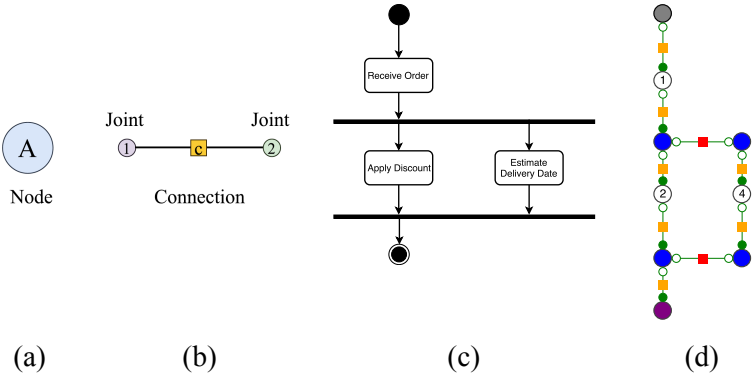


Figure 7.2: ImitGraph notation: (a) a node. (b) a connection with two joints. (c) a sample activity diagram. (d) an ImitGraph equivalent to the activity diagram.

Different types of joints, connections and nodes should be defined for an experiment. This includes specifying properties such as orientation of the connections (vertical, horizontal or any), the connection types each node accepts, the joint types of each connection, and the anchor point from which a connection can be attached to a node (left, right, top, bottom, any combination of these or any). ImitGraphs distinguish types by color. A proper definition of components allows ImitGraphs to imitate a graphical model. Figures 7.2c and 7.2d show a simple activity diagram and its ImitGraph equivalent. Larger elements of the graphical models

are simulated by multiple ImitGraph nodes (e.g., fork and join elements of the activity diagram).

7.4.1 Tool implementation

Tablets are becoming increasingly popular as devices for creating and viewing requirements models: they are mobile, can be used in the stakeholders' work context [MSG+07] and can also be connected for collaborative modeling [WSG15]. However, the smaller the display screen, the more challenging navigation in models becomes [GG17a]. Therefore we decided to target medium-sized tablets (with a 10-inch screen) in order to investigate the extent to which the navigation technique used influences the performance of users when modeling requirements.

Based on the goal of this study, we specified the following requirements for our tool. (i) Supports creating and editing ImitGraphs. (ii) Supports both classic ZoomScroll and FlexiView for navigation. (iii) Is responsive and highly interactive so that it does not affect the performance of the users. (iv) Allows integration of a physics engine in order to achieve the realism we needed. (v) Is able to show smooth animations. (vi) Is able to access low-level interaction functions to record fine details of user interactions. (vii) Has a user interface similar to common modeling tools.

We decided to use the LibGDX game engine [Oeh13] for developing our tool to have adequate performance, responsiveness and interaction functions. Although we did not seek high accuracy in

our physics simulation, we used the Box2D physics engine [Par13a] which helped us to achieve realistic movements of FlexiView's magnets and springs. Accessing low-level interaction functions allowed us to precisely record detailed interactions of the users from within the tool. The output dataset contained single entries for actions such as selecting a node and multiple entries for continuous actions such as dragging a node, both with millisecond precision.

In order to implement semantic zooming, we defined ImitGraphs to hold three different textual values named title, abstract and description in each node. Based on the size of the node on the screen and according to our defined thresholds the tool displays (i) first character of the title, (ii) title, (iii) title+abstract, or (iv) title+abstract+description.

Figure 7.3 shows the user interface of our tool. The central part is the graph editing area. The tool provides both ZoomScroll and FlexiView navigation in this area. For the purpose of our experiment, we included a hidden option to selectively disable one of them. When a node or connection is selected, its context menu appears above it. From the context menu, the users can delete the selected object, open a text editing window and create a magnet. When a node is magnetized, a slider appears at the bottom for setting the magnet's strength. For each magnet, a button is displayed in the magnet pane so that the user knows how many magnets exist and can select one of them easily by pushing its corresponding button. From the node menu and connection menu, the user can drag a node or connection and drop it on the editing area. The file menu contains buttons for creating/opening/saving

files, undo and redo. The task menu contains the button that supervisors use for loading tasks, a timer and a finish button that users push when they finish the task.

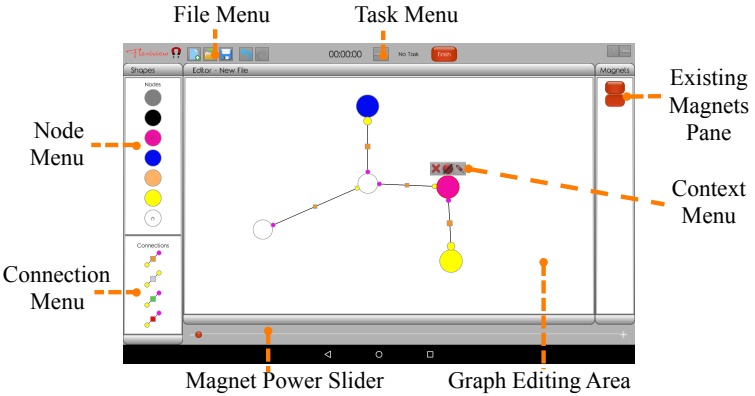


Figure 7.3: User interface of the tool used in the experiment.

7.5 Experiment Tasks

We designed the experiment tasks based on four common operations that are regularly performed when working with RE artifacts [GG16]: (i) Searching for specific information, (ii) Editing an artifact using information from other parts of the same artifact, (iii) Searching in a hierarchical structure, (iv) Searching and matching information stored in more than one artifact.

In addition, we considered two criteria. First, the tasks need to be challenging enough so that we can observe time and error

differences. Second, the tasks have to be doable within the limited time frame of a trial. The latter limits the size of the models used in our experiment. Based on these considerations we designed the following four tasks.

7.5.1 Task Search

In this task, the participants are given an ImitGraph that mimics a class diagram, i.e., nodes represent classes and edges represent associations. Figure 7.4 shows this graph and a magnified part of it. Each node holds a class specification consisting of a name, a list of attributes and a list of functions. These become visible when zooming into a node. The participants must find all nodes that contain an attribute named "id" (which is present in five out of 30 nodes) and add a function named "getId()" to these nodes.

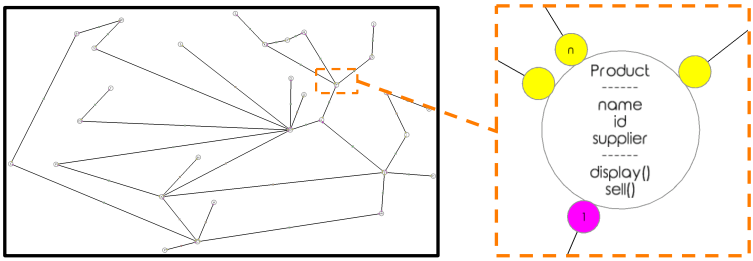


Figure 7.4: The ImitGraph used in task Search, representing a class diagram. The magnified part shows a node with the class name, attributes, and functions.

In this task, we wanted to observe how the participants search the whole model and how efficient their search paths are with regard to missed nodes and repeated visits.

7.5.2 Task Recreate

In this task, the participants are given three ImitGraphs in one artifact (Figure 7.5). The left graph represents a given UML activity diagram. The right graph represents an activity diagram under construction, where the already existing parts have been copy-pasted from the left graph.

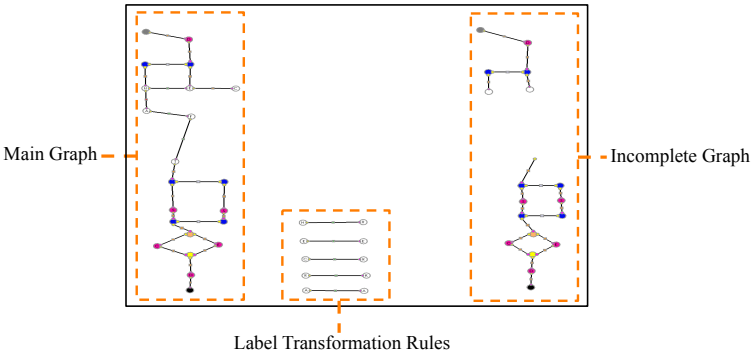


Figure 7.5: The artifact that was used in task Recreate with its three ImitGraphs.

The participants must now complete the missing parts of the right graph such that (i) the two graphs are structurally identical and (ii) the names of the re-created nodes are transformed according to the rules specified by the ImitGraph in the middle. The transformation

works as follows: when participants create a new node in the right graph, they have to determine the label n of the corresponding node in the original (left) graph. Then they have to determine whether there is a transformation rule for n, i.e., a connection in the middle ImitGraph that has a left node labeled n. If this is the case, the newly created node in the right graph gets the label n' which is given by the right node of the transformation rule. Otherwise, the new node is labeled n.

In this task, we wanted to observe how efficiently the participants navigate between three known locations of the artifact and to what extent the off-screen information that they need affects their performance.

7.5.3 Task Hierarchy

In this task, the participants are given an ImitGraph with an embedded hierarchical structure. This ImitGraph mimics a structure found in i* models, which may contain hierarchical goal decompositions, but also contain other links between the nodes of the model. In this ImitGraph, the edges expressing hierarchical structure are marked with a small green rectangle in the middle of the connection, while the other edges are labeled red or orange. The root node of the hierarchy that the participants had to explore is labeled *Start*.

Figure 7.6 shows the given graph and the embedded hierarchical structure. Hierarchy edges link the nodes on the first decomposition

layer with the root node. Every node on the first decomposition layer may have hierarchy edges to nodes on the second decomposition layer, etc. The hierarchy is not visible in the spatial arrangement of the nodes, which is also typically the case in i* models.

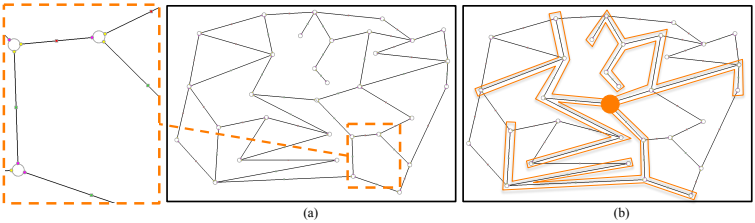


Figure 7.6: (a) The ImitGraph given to the participants and an enlarged region. (b) The hierarchical structure is highlighted (not shown to the participants).

The task of the participants is to traverse the hierarchy, starting from the root node, and delete all occurrences of the attribute *static* from the nodes belonging to the hierarchy, but not from any other nodes.

This task was designed to observe how participants traverse a hierarchical structure and how well they remember the relative location of nodes that they have already visited.

7.5.4 Task Matching

While we designed the first three tasks for comparing the performance of ZoomScroll vs. FlexiView navigation in a single artifact,

this task was designed to explore the effects of FlexiView navigation in a single artifact vs. presenting the same information in multiple artifacts (with ZoomScroll navigation). For this, we defined two settings. In the first setting, the participants are given an ImitGraph with three central nodes which are connected to five secondary nodes each (Figure 7.7).

Each central node contains six central codes and each of the secondary nodes contains one secondary code.

In the second setting, the participants are given four files. The first file contains the three central nodes. The second, third and

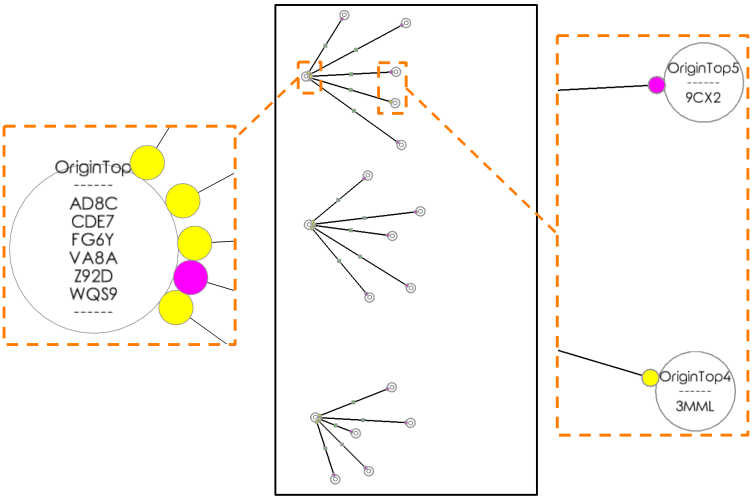


Figure 7.7: The artifact used in the first setting of task Matching. A central node and two secondary nodes are magnified, showing the codes they contain.

fourth files contain the five secondary nodes that are connected to the first central node, second central node and third central node, respectively. The names of these files indicate to which central node they belong. As usual on tablets, the participants can open and view only one file at a time.

In both settings, the task of the participants is to inspect the linked pairs of central and secondary nodes and identify those that have matching codes. A match occurs when the first character of the code in a secondary node is equal to the last character of one of the six codes in the corresponding central node. For example, in Figure 7.7, the code "9CX2" in secondary node *OriginTop5* matches the code "WQS9" in central node *OriginTop*. In this task, the participants were allowed to use a small piece of paper (1.5×5cm) for taking notes.

This task was designed to observe whether presenting related information in a single graphical model with semantic zooming as offered by FlexiView has benefits over presenting the same information in multiple related files.

7.6 The Experiment

7.6.1 Participants

We had two criteria for recruiting the participants: (i) being familiar with graphical requirements models, (ii) knowing how to

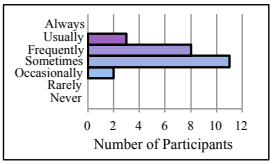
use a tablet. We particularly targeted students who had recently taken the Introduction to RE course offered at our university. Advanced knowledge in RE was no requirement.

We recruited 24 students (12 advanced undergraduates and 12 graduate students) from the student population in Computer Science at our university. Each participant received an honorarium equivalent to about $ 20 for participating in the experiment. The age of the subjects ranged from 21 to 44; the mean age was 26.70 years. Figures 7.8a and 7.8b show the participants' experience in using graphical models and tablets based on their claims. All had normal or corrected to normal vision and did not suffer from any form of color blindness.

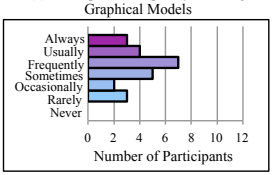
7.6.2 Experiment Design

After designing the tasks, we performed two pilot studies with a postdoc and a PhD student who were knowledgeable in this field, but not involved in this project directly. We adjusted the task difficulties and time limits based on the pilot results, thus ensuring that all tasks were doable in the given time. We used time limits for motivating the participants to work as fast and productively as they would have to work on real tasks in industry. An on-screen timer showed the remaining time during the tasks. The participants were asked to finish the tasks even after the timer went off.

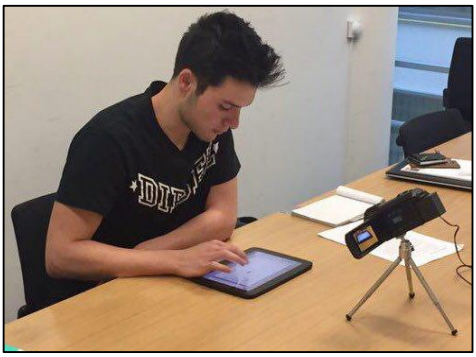
All experiment trials were carried out on a 10-inch Android tablet. The whole sessions were recorded with a video camera such that



(a) Participants' Frequency of Using Graphical Models



(b) Participants' Frequency of Using Tablet



(c) Experiment Setup

Figure 7.8: Experience of users in using (a) graphical models and (b) tablets, (c) Experiment Setup

only the tablet screen and the hands of the participants were recorded. We used these videos only as a complement to the data recorded by our tool: they helped us resolve ambiguities and double-check special cases, e.g., when a search path crossed a node but the participant did not visit it. Figure 7.8c shows the experiment setup.

Each experiment trial comprised a tutorial session, four tasks, and a final survey. Each trial started with a short introduction of our tool followed by a training session. During the training session, we first made sure that the participants learned how to use the basic functions of the tool (e.g., creating nodes, connecting nodes and moving them). Then, we explained how FlexiView works and guided them to do basic operations (e.g., creating a virtual magnet, changing a magnet's strength and managing the magnets). We also explained how they could zoom and scroll. Next, we shortly

explained the concept of ImitGraphs. At the end of the tutorial, we gave the students about five minutes to try out the tool. The mean tutorial time was 15 minutes and 20 seconds.

Each participant executed the first three tasks two times: one-time using ZoomScroll and the other time using FlexiView. They executed the fourth task, once using the MultiFile+ZoomScroll technique and once using FlexiView. We equally divided the participants into two groups: FlexiView-first and FlexiView-second. The order of the tasks was the same for both groups of participants, but the FlexiView-first participants executed each task first using FlexiView then the other technique; FlexiView-second participants executed each task first using the other technique and then FlexiView. We asked the participants to mark their satisfaction with the ease of use after each execution (i.e., two times for each task).

At the beginning of a new task, we provided the instructions about the task on a sheet of paper and gave the participants time to read and understand them. The supervisor answered their questions about the task. When they felt ready, the supervisor started the on-screen timer for the task and the participant had to finish it before the timer went off. When they were done, the participants pressed the *Finish* button in the menu bar. The participants were asked to finish their task even after the timer went off. After the participants finished all four tasks, we asked them to fill out the final survey. This survey contained eight statements in four groups: learnability, memory usage, performance and overall satisfaction.

The experiment was conducted in January and February 2018. Each trial lasted approximately one hour and 35 minutes. The descriptions of the tasks, the training material, and the satisfaction questionnaires were all in English[1]. We chose a quiet place for running the experiment and made sure that no disturbance occurred during the trials. All experiment trials were supervised by the first author to gather consistent data.

7.7 Results and Discussion

In this section, we report our results of analyzing the gathered interaction data for the four tasks. Since we designed our tool to record user interactions in fine details, the resulting raw data set (more than 26 hours of recorded interactions) was so large that we could not analyze it without postprocessing. We developed a Java program to extract higher level data from the raw data, e.g., visited nodes, the speed of scrolling, states, and duration. Then, instead of the raw data, we analyzed the extracted data to find the information that we were looking for, e.g., missed nodes, repeated visits, and searching paths.

We used two types of charts for depicting our results: box plots and bar charts. Box plots provide a visual image of the distribution of the data, while bar charts allow comparing populations of different answers on a Likert scale. Since we repeatedly compare two datasets, one from FlexiView and one from ZoomScroll, we

[1] `https://re18flexiview.page.link/jdF1`

employed a paired two-tailed student's t-test [TA13] to find if the differences are statistically significant. Since all t-tests were done with results from 24 participants, the degree of freedom was 23. Also, we set 0.05 as Alpha for all t-tests, i.e., we used p-values of 0.05 to determine if the differences were significant. On the right side of the box plots, mean and standard deviation of each execution and t-test results (t-value and p-value) for each task are presented. We refer to each execution of a task as Task+Technique, e.g., Search+ZoomScroll. We present our results in four categories: efficiency, effectiveness, cognitive load, and satisfaction. The first two categories answer RQ1. The third and fourth categories answer RQ2 and RQ3 respectively.

7.7.1 Efficiency (RQ1)

Completion time is the main factor that is taken into account for assessing the efficiency. Figure 7.9 shows the distribution of completion times for all tasks and techniques. In all tasks, the mean of completion time for ZoomScroll executions is higher than for FlexiView executions. In tasks Search, Recreate and Matching the difference is significant ($p < 0.05$). In task Hierarchy, the difference is not significant ($p > 0.05$).

To find out why FlexiView was not significantly better than ZoomScroll in task Hierarchy, we studied the videos of this task and found that with ZoomScroll, this task could be accomplished without much zooming in or out: the participants could zoom in on the

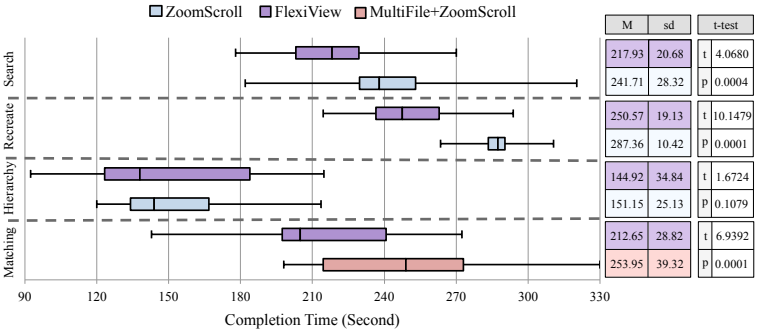


Figure 7.9: Completion times for four tasks

root node once and then find their path without further zooming, using the hierarchical links as guides.

The navigation technique is one of the many parameters that affect the completion time. In our study, other parameters (e.g., typing skill on a touch-screen keyboard) act as noise. We managed to reduce the noise for two tasks (Search and Recreate). First, we identified four states into which the participants switched in each task. Second, we classified each point of the timeline into one of the four states. Finally, we aggregated the times that the participants spent in each state.

We identified the following four states in task Search: (i) *Orientation*: In this state, users decide about the search path and the next node to visit. The characteristics of this state are that the zooming level is not high enough to check the details and the user does not scroll. (ii) *Navigation*: In this state, users know the next node they want to visit and try to reach it. The characteristic of this state is

that users change the zoom level and/or scroll. (iii) *Process*: In this state, users check the content of the nodes and decide whether to change it or leave it. The characteristic of this state is that the zooming level is high enough to check the details of the nodes and users do not navigate. (iv) *Edit*: In this state, users open the node editor and change the content of the node.

In the process of classification, for deciding whether the user is scrolling or not, we defined a speed threshold. When the user moves faster than the threshold, we classified it as scrolling; Otherwise, we classified it as not scrolling. Figure 7.10 shows how much time users spent in each of the four states. Editing and processing times were generally in the same range for both techniques with no statistically significant difference. The main reason that participants completed this task faster in FlexiView executions lies in the differences in orientation and navigation times which are significantly different based on the t-test results.

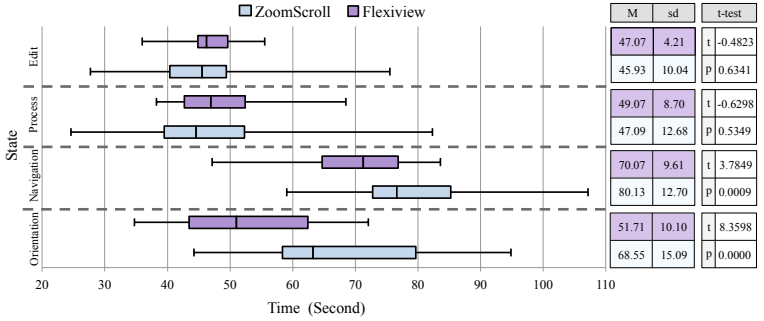


Figure 7.10: Search Task: consumed time in each state

In task Recreate, unlike task Search, the participants knew the locations of the information they needed. They needed to constantly move between three sections of the canvas. The participants were in one of the following four states during the whole task: (i) processing the main graph: when they studied the structure and the content of the nodes of the main graph, (ii) processing the rules graph: when the participants checked the label transformation rules, (iii) navigation: the time that they spent to move the focus between the three sections, and (iv) creation: the time that they spent to create the required nodes and connections.

We identified navigation states similar to the previous task. Distinguishing the rest of the states was done based on the location of the focus. Figure 7.11 shows how much time the participants spent in each state. Among the four states, navigation time has the highest difference and was the main reason why the participants finished this task using FlexiView in a shorter time.

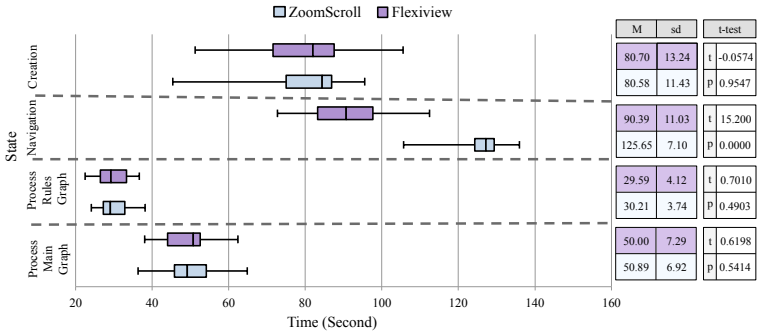


Figure 7.11: Recreate task: consumed time in each state

In summary, in tasks Search and Recreate FlexiView affected the efficiency positively. In task Hierarchy, the results of FlexiView were comparable to ZoomScroll. The results of task Matching suggest that showing connected files as a graph and navigating it by FlexiView is more efficient than opening and viewing the files separately.

7.7.2 Effectiveness (RQ1)

In tasks Search, Recreate and Hierarchy, the number of errors that participants made in their final results was insignificant for both techniques. However, the participants made errors that did not affect their final results. For example, in task Search, some participants missed visiting some nodes. We call this type of errors navigation errors. Navigation errors could potentially result in errors in the final results. To compare the effectiveness of FlexiView and ZoomScroll, we took the navigation errors into account. Figure 7.12 shows the distribution of navigation errors for Search+ZoomScroll and Search+FlexiView. We defined three types of navigation errors for task Search: (i) the number of nodes that were missed to be visited, (ii) the number of repeated visits, and (iii) the number of times that the search path intersected itself. A node is counted as visited when the user is not zooming, the scrolling speed is below the threshold that we defined, and the semantic zooming level allows to see the content of the node. We drew a search path based on the sequence of the nodes that each user visited and counted the number of times this path intersected

itself. This will be discussed in more detail in Section 7.7.3. The mean numbers of errors for FlexiView executions are lower and differences between FlexiView and ZoomScroll are statistically significant ($p < 0.05$).

Figure 7.12: Search task: the number of errors of different types

We defined three types of errors for task Hierarchy: (i) the number of nodes that were missed to be visited, (ii) the number of repeated visits, and (iii) the number of visited nodes that were not supposed to be visited. Figure 7.13 shows the distribution of these errors. FlexiView executions have a lower number of errors and the differences are statistically significant for missed nodes and wrong visits ($p < 0.05$). The p-value of redundant visits is slightly over the 0.05 threshold.

In task Recreate, the participants needed to visit the main graph and the label transformation rules graph multiple times. Some of these visits were necessary and some were not. The necessity of the visits depends on the participants' strategy and how much information they can keep in their mind. Distinguishing them is not trivial and we cannot make any conclusion about the difference of effectiveness in this task.

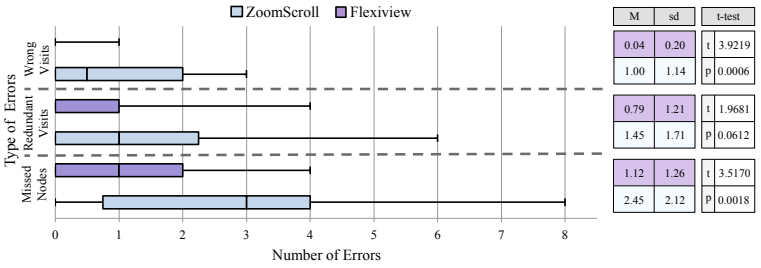


Figure 7.13: Hierarchy task: the number of errors of different types

In task Matching, we counted the mistakes that participants made in the list of matched codes that they handed in. The errors comprise missed matches and wrong matches. Figure 7.14 shows the distribution of total errors for both techniques in tasks Search, Hierarchy, and Matching. In all these three tasks, the participants made fewer errors using FlexiView and the observed differences are statistically significant ($p < 0.05$).

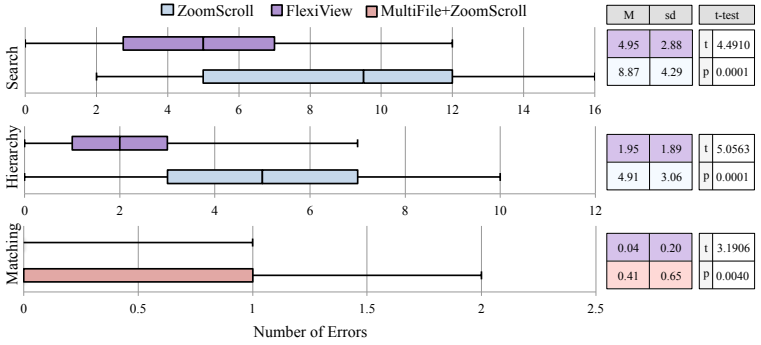


Figure 7.14: Total number of errors for tasks Search, Hierarchy and Matching

In ZoomScroll, users have to zoom in when wanting to view details. This enlarges the model canvas, which means that some parts of the model disappear from the screen. In this situation, users most likely lose the overview of the artifact, their current position and the relative position of the nodes. This results in missing some nodes, redundant visits of some other ones and visiting nodes that are not needed to be visited. When using FlexiView, users can check the details while keeping an overview of the information on the screen. The cost of showing details and keeping an overview at the same time is distortion, which is caused by non-proportional scaling. Based on our results, despite the introduced distortion, FlexiView reduces the number of navigation errors.

7.7.3 Cognitive Load (RQ2)

When zoomed in, if the users need an overview of the artifact (e.g., for planning the search path), they either zoom out to have the whole artifact back on the screen or they rely on their memory. We were interested in knowing the characteristics of the search paths of the participants when using FlexiView and ZoomScroll in task Search. The path that participants traversed provides insight into how well they could plan based on the overview of the artifact they had on the screen and/or in their mind. Figure 7.15 shows the resulting search paths for two participants (p4 and p17). p4 was in the FlexiView-second group and p17 was in the FlexiView-first group. To compare the search paths we decided to use a simple measure: the number of times each search path

intersected itself. Figure 7.12 shows the distribution of the number of intersections for ZoomScroll and FlexiView. The mean number of errors show that the search paths of FlexiView executions had fewer self-intersections. We can also visually observe that they were more regular and had sweeping patterns. The t-test results show that the difference is significant.

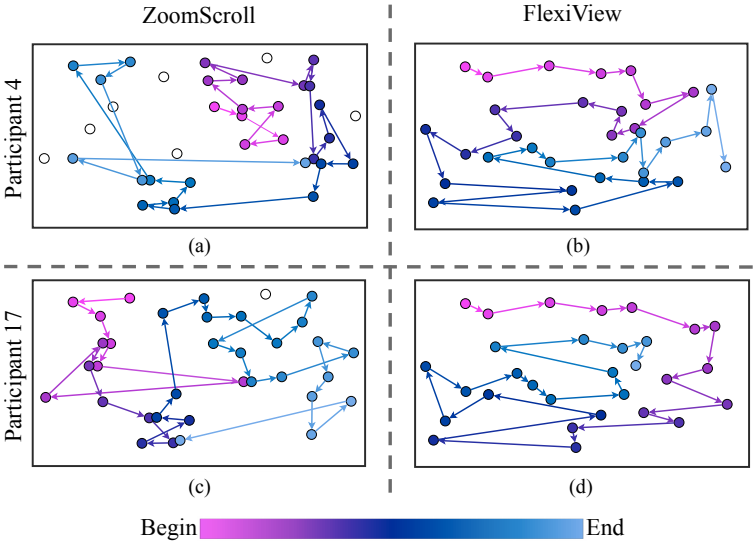


Figure 7.15: Samples of search paths: (a) and (c) ZoomScroll, (b) and (d) FlexiView. The color shows the sequence and the white nodes were missed.

In task Matching, we gave small pieces of paper to the participants so that they could take notes if needed. Writing notes on these papers was an indicator of how much they needed to use their memory. We assumed that when the information that people need to keep in

mind is above some threshold, they write more notes. We deliberately gave the participants a small piece of note paper only, so that they had to use their memory and could not write down all needed information. Figure 7.16 shows eight samples of note papers of four participants in MultiFile+ZoomScroll and FlexiView executions. To compare the note papers of the participants, we counted the number of letters they wrote on their MultiFile+ZoomScroll papers ($M = 27.5$, $std = 16.95$, $min = 18$, $max = 72$) and on FlexiView papers ($M = 2$, $std = 3.06$, $min = 0$, $max = 10$). The t-test results shows a significant difference ($t = 7.1618$, $p < 0.0001$).

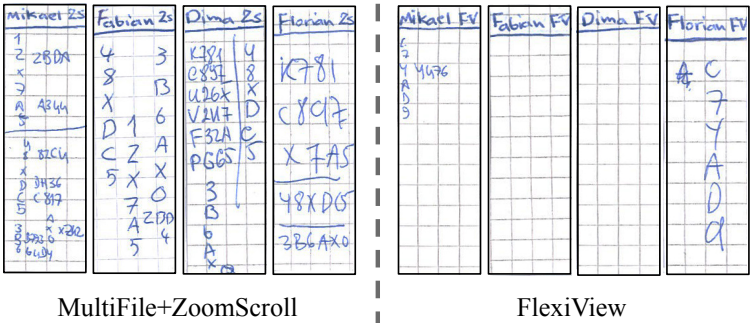


Figure 7.16: Samples of note papers used by four participants when using FlexiView and MultiFile+ZoomScroll

Another observation in task Recreate was that 91.6% of the participants (22 out of 24) used multiple magnets to magnify the main graph and the rules graph at the same time to see more relevant data on the screen. Figure 7.17 shows a snapshot of such a situation. Although using more magnets causes more distortion, the participants exploited the advantage of having multiple foci.

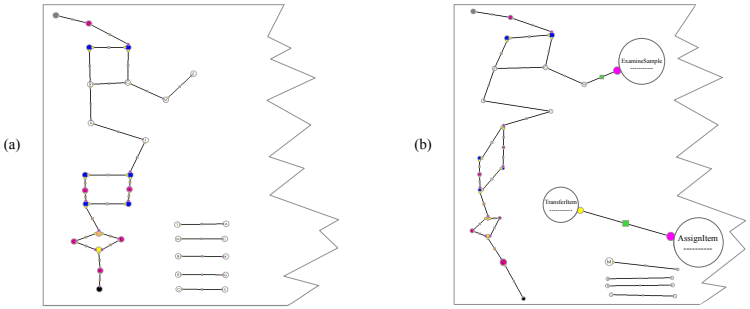


Figure 7.17: Two partial snapshots of the artifact used in task Recreate: (a) normal and (b) with three magnets

7.7.4 Satisfaction (RQ3)

After each task execution, the participants rated their satisfaction with the ease of use of the navigation technique they had used for that task on a Likert scale. Figure 7.18 shows the result of these surveys. The first and second column show the results of ZoomScroll and FlexiView executions respectively. The third column shows the difference between ZoomScroll and FlexiView answers. For comparing the answers, we avoided quantification which could exaggerate or understate the difference. Instead, we compared the answers separately by calculating the differences and showing them in a different bar chart.

Satisfaction is a complex feeling which is influenced by different factors. The purpose of the final survey was to collect more clues about this feeling. In this survey, we asked the participants to rate how much they agreed with eight statements in four different

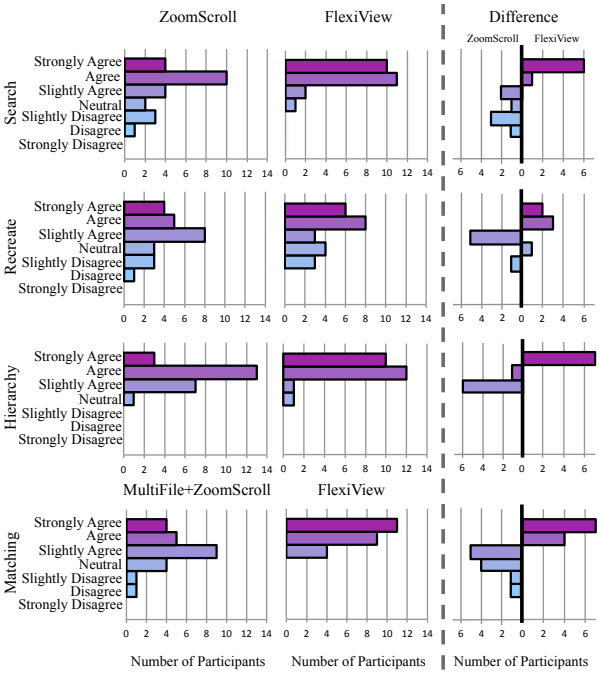


Figure 7.18: Satisfaction with ease of use of the used technique after each execution. The third column depicts the difference.

topics: learnability, memory usage, performance, and ease of use. Figure 7.19 presents the ratings for these eight statements, labeled S1-S8.

Learnability

In comparison to common navigation techniques used in commercial tools, FlexiView is more complex. However, we observed

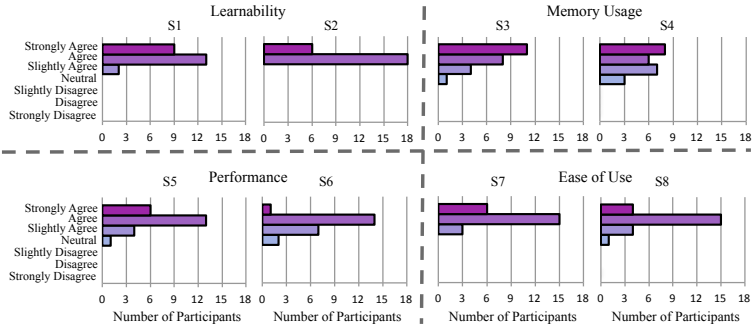


Figure 7.19: Final survey results: S1: I learned to use FlexiView quickly. S2: Other people will learn to use FlexiView quickly. S3: Users have the overview of the artifact more often by using FlexiView. S4: Users are more aware of their position inside the artifact by using FlexiView. S5: Users become more productive by using FlexiView. S6: Users make fewer errors by using FlexiView. S7: I recommend FlexiView to be implemented in other tools. S8: I am satisfied with the ease of using FlexiView.

that all participants learned to use it during a short tutorial ($M = 15.33, min = 12.20, max = 21.75$) and developed their skill during the tasks. The result of the final survey confirms that the participants learned to use FlexiView quickly (S1) and believed that others would also learn it quickly (S2). Based on our observations, the participants were excited to see a physics-based user interface. The physics simulation made the transitions between distorted views of the artifact smooth and predictable. All participants successfully used FlexiView alone for navigating.

Memory Usage

We discussed some measurements of user cognitive load, e.g., memory usage. Since exact measurement of memory usage is not possible, we asked the participants how much overview they had while using FlexiView (S3) and how aware they were of their position (S4). The results show that FlexiView was successful in both.

Performance

Although we measured the performance of the participants based on the collected data, we still were interested to know how the participants felt about their performance. They agreed that they were more productive using FlexiView (S5) and made fewer errors (S6).

Ease of Use

Finally, we wanted to know how satisfied the participants were with the ease of using the new navigation technique that they were trying for the first time. We asked if they were satisfied with the ease of use of FlexiView (S8) and if they recommend it to be implemented in RE modeling tools (S7). The high level of user satisfaction was an aggregated result of all the other aforementioned factors.

7.8 Threats to Validity

In this section, we discuss the usual four categories of threats to validity [WRH+00]. Measurements affect the *Conclusion Validity*. Although we recorded the interactions very precisely, the information extraction was based on thresholds that we defined (e.g., the speed threshold which distinguishes between visiting a node from hovering over it). We addressed this in two ways. First, we did the analysis with different thresholds and the extracted data was not sensitive to these thresholds. Second, we double-checked the special cases (e.g., missed nodes on the search path) with the recorded videos. In addition, in order to have equal motivation for finishing the tasks as soon as possible, we put a count-down timer on top of the screen. To have unbiased statements in the surveys, we avoided making direct comparisons between two navigation techniques. However, FlexiView being the subject of some of the questions may cause bias.

For *Internal Validity* we used different artifacts for ZoomScroll and FlexiView executions to remove the learning effect. The artifacts were basically similar with slightly different layouts and node contents. We made sure that the critical properties such as the number of nodes, number of answers, branching factor and the ratio of relevant/irrelevant information were the same. Additionally, we did not imply any expectation and used comparative questions only after all tasks were done to avoid any bias towards FlexiView. Furthermore, we compared the results of the FlexiView-first and FlexiView-second groups and did not find a significant difference.

The difference between ZoomScroll and FlexiView is a threat to *Construct Validity*. Zooming and scrolling have been matured during decades of use. The users of software tools have a great deal of experience in using them. FlexiView is new, more complex and immature. Since most of the results of this study are in favor of FlexiView, overcoming this threat (e.g., by improving the implementation and longer tutorial sessions) would only change the intensity of the results.

We recruited 24 students for our experiment which is a threat to *External Validity*. We addressed this in two ways. First, we used ImitGraphs instead of real requirements models to decrease the influence of previous knowledge on the results. Second, the tasks comprise basic common operations that are typically done on graphical requirements models. However, we do not claim that this study is a proof of how much FlexiView improves the effectiveness, efficiency and user satisfaction of requirements modeling tools. Nevertheless, this study shows a great potential for such a UI technique for improving requirements modeling tools. Using ImitGraphs introduces another threat. Although the behavior of the users when using ImitGraphs is similar to their behavior when using graphical models [Lay18], it is still possible that the results of this experiment will be different when using RE graphical models instead of ImitGraphs.

7.9 Conclusion and Future Work

In this study, we compared the effect of two navigation techniques on the performance of working with RE modeling tools: ZoomScroll, a traditional, widely used technique, and FlexiView, a new focus+context technique. The comparison was done using the precise interaction data recorded by our tool during the experiment designed for this study. For the experiment, we recruited students with some RE knowledge to perform basic operations that are usually done on RE models, once using ZoomScroll and another time using FlexiView.

Using FlexiView, the participants finished three tasks out of four in a shorter time and made fewer navigational errors in all tasks. They had a better overview of the artifacts when using FlexiView which resulted in better searching paths which in turn resulted in shorter completion time and fewer errors. We observed that the participants were excited about using the physical metaphors of FlexiView. They learned it quickly and expressed their high satisfaction with its ease of use. In conclusion, we discovered a great potential in a navigation technique such as FlexiView for improving the effectiveness, efficiency and user satisfaction of RE modeling tools.

This work is just a beginning of a path which ultimately will result in navigation techniques that are highly tailored to RE modeling needs. The next steps will be experimenting with various navigation techniques using our platform and comparing them

with more complex baselines, focusing on the challenges of RE modeling tools individually, experimenting with more specific tasks including larger models, experimenting on devices with different screen sizes and incorporating modern human-computer interaction input methods.

Acknowledgement

We would like to thank all the participants for their time and valuable contributions that made this research possible.

Chapter 8

Conclusion

Requirements engineers dedicate a significant part of their time to working with requirements artifacts using software tools. However, the usability challenges that they face when creating, viewing and updating these artifacts are not empirically studied. In this thesis, we conducted an exploratory study to find such challenges and the characteristics of artifacts that influence those challenges. Based on the identified challenges, we created a new navigation technique tailored to requirements artifacts. Finally, we evaluated our navigation technique experimentally by comparing its performance to a classic navigation technique. In this chapter, we first revisit the research questions that we introduced in Section 1.3. Then, we check the thesis statement to see how much it is supported by the material presented in this dissertation. Finally, we picture how this research may continue in the future.

8.1 Revisiting the Research Questions

RQ1: What challenges do practitioners face when navigating inside requirements artifacts and which characteristics of artifacts do influence these challenges?

In Chapter 2, we presented our exploratory study, including 29 interviews and 42 survey responses, on characteristics of requirements artifacts and the challenges of working with them. We found that a significant number of artifacts does not fit on the users' screens and using larger screens cannot solve this problem. In addition, the practitioners work with multiple artifacts at the same time and work on artifacts collaboratively without using dedicated tools. We also found that the practitioners face challenges when working with large artifacts, e.g., they have to rely on their memory and maintain an overview of the artifacts manually. In working with interconnected artifacts, the practitioners face challenges too, e.g., they have to switch between tabs and work in small windows. The practitioners use different workarounds to tackle the challenges that they face, e.g., they use paper and whiteboards instead of software tools to have more speed, flexibility, and space. Their workarounds are time-consuming, inefficient and error-prone in comparison to using dedicated features in requirements tools.

RQ2: How can we devise a new navigation technique to tackle the usability challenges that practitioners face when working with RE artifacts?

We encountered keywords such as “overview of the artifact” and “memory usage” recurrently in the interviews of our exploratory study. The reason was that zooming in on large artifacts and opening artifacts in multiple small windows caused parts of the artifacts to be located outside the visible area. This resulted in losing the overview and being forced to rely on the memory for the off-screen parts. The problem of presenting large documents has already been investigated and techniques such as focus+context have been introduced for this purpose. The goal of focus+context techniques is to keep as much information as possible on the screen by reducing the details of the less relevant parts. This helps their users to continuously have an overview of the document which in turn leads to lower cognitive load. The goal of focus+context techniques was promising for addressing the challenges we identified in working with requirements artifacts. Therefore, we designed a focus+context technique named FlexiView specifically for requirements artifacts.

Despite the aforementioned advantage of focus+context techniques, they have a side effect known as distortion when keeping the size of the artifacts constant (i.e., the whole artifact is visible without zooming and scaling). In order to tackle the distortion in FlexiView, we used physics metaphors. Since the human mind is familiar with physics phenomena, using physics affordances makes the distortion more predictable and thus more tolerable. A conceptual description of FlexiView is presented in Chapter 3. We provided more details of FlexiView in Chapter 7 where we report its implementation in our experimental tool.

RQ3: How can we develop an experimental tool that allows side-by-side comparison of different navigation techniques in RE artifacts?

For evaluating FlexiView, we needed a platform that could test different navigation techniques fairly and produce generalizable results. For having fairness, we decided to implement different techniques in one environment. Therefore, we created an experimental tool which supported FlexiView, zooming and scrolling. For having generalizability we made the experimental tool (i) to be similar to existing requirements modeling tools and (ii) to support the ImitGraph notation. To make our tool similar to the existing tools, we conducted a market study that is presented in Chapter 5. In this study, we examined ten touch-screen requirements modeling tools and made a list of their common features. Instead of supporting many graphical notations for producing generalizable results, we created the ImitGraph notation which is presented in Chapter 4. ImitGraphs can imitate the behavior of different node-and-edge graphical notations. Finally, we implemented a tool with the requirements that we found in the market study. Our tool is a modeling tool for ImitGraphs and supports FlexiView, zooming and scrolling. Additionally, it is capable of logging user interactions precisely for further analysis. Our tool is presented in Chapter 6.

RQ4: To what extent does a physics-based focus+context navigation technique affect efficiency, effectiveness and user satisfaction of working with artifacts?

Finally, we used our experimental tool to evaluate FlexiView. This evaluation is presented in Chapter 7. We conducted an experiment with 24 participants in which they were asked to perform tasks once using FlexiView and another time using zooming+scrolling. Afterwards, we analyzed the precise data recorded by our tool. We found that a significant portion of the time in each task is spent for planning and navigating in both techniques. This finding is another reason why improving navigation techniques in requirements modeling tools can improve the performance of requirements engineers tasks. The data showed that FlexiView significantly reduced the time that users need for planning which resulted in shorter completion times in most of the cases in comparison to zooming+scrolling. By examining the errors that users made, e.g., missed nodes and redundantly visited nodes, we found that having an overview of the artifact helped the users to plan more effectively with fewer errors. The effect of having an overview of the artifact was also evident in the search paths of the users, which we visualized using the recorded data. The search paths in FlexiView trials were regular and had sweeping patterns. All our findings show a great potential in FlexiView for improving the usability of navigation in requirements artifacts.

8.2 Revisiting the Thesis Statement

In the beginning of this research, we found that there is a significant number of large and interconnected artifacts and practitioners face challenges in working with them since they do not have tools made

for coping these challenges. Based on the existing successful experiences of designing navigation techniques, we created a navigation technique specifically for requirements modeling tools to tackle the challenges we found. We created a framework which allowed fast and fair comparisons of navigation techniques while preserving the generalizability of the results. Finally, we evaluated our navigation technique in our experimental tool and found a great potential in it for enhancing the performance of the requirements engineers in tasks that involve artifacts.

8.3 Future Work

In this thesis, we compared FlexiView and zoom+scroll. The results of this comparison show that FlexiView has improved both the usability factors and the performance of its users. This improvement is a proof of concept that usability of RE tools affects the performance of working with requirements artifacts, and dedicated navigation techniques can alleviate a part of the usability challenges in requirements modeling tools. This is just the beginning of a path that will ultimately result in more efficient and effective requirements tools that satisfy their users more than today's tools do. We divided this thesis into three interwoven tracks that we followed: (i) finding the usability challenges of working with requirements artifacts, (ii) making usability tests of requirements tools faster and cheaper, and (iii) creating a dedicated navigation technique for requirements artifacts. In this section, we discuss how each of these tracks can continue in the future.

The first step in improving the usability of requirements modeling tools is to identify the existing challenges and their root causes. We used interviews and survey responses for this purpose which provided us with a high-level picture. This picture was valuable to our study since the literature did not provide us with any clue in this matter. The lack of perspective of what we had to expect led to a wide search domain. We chose exploratory research methods for this purpose such as interviews and surveys. Even though we achieved a high precision of measurements that is possible in these methods, the precision can be enhanced in a future research with a different research method. We found statistics about the characteristics of artifacts, screens, and tools, and found a number of challenges of working with large and interconnected artifacts. Nevertheless, we cannot claim that our results were comprehensive. Now that we have a big picture, we can concentrate the focus on specific characteristics and challenges, and choose research methods that allow more precise measurements. For example, we found that requirements engineers use large artifacts by measuring how much of their artifacts fit on their screens. A more precise measure of the size and information density of the artifacts will guide the designers of requirements engineering tools to know the cause of usability challenges, their importance and how to tackle them. Similarly, we can design studies for the interconnectivity of the artifacts, the number of simultaneously used artifacts, the significance of viewing overviews of the artifacts, the amount of cognitive load, and so on.

Enhancing the usability of requirements tools requires persistent cycles of testing and improvement. After finding a challenge, its

cause and a solution to it, the solution should be evaluated. Usability evaluations are done by comparing a new solution to another solution or to another version of the same solution. In order to be able to conduct an adequate number of usability tests, we should minimize their cost. At the same time, we should increase their accuracy and the generalizability of their results. In this thesis, we introduced ImitGraphs and created an experimental tool using them. The ImitGraphs concept is in its initial development stage. It needs more research to become a graphical notation that can effectively represent other requirements engineering notations in usability tests. We have used ImitGraphs for representing activity diagrams, class diagrams, and goal decomposition models. More diagrams should be tested and additional properties should be added to ImitGraphs when needed. Furthermore, our experimental tool can be developed to support more common features of requirements tools in order to broaden the range of test cases. Furthermore, we need intermediate tools to preprocess raw data and provide higher-level information for the final analysis.

We designed and tested FlexiView, our dedicated navigation technique for requirements artifacts. FlexiView showed a great potential for enhancing the usability of requirements modeling tools. We found its strengths and weaknesses while we were experimenting with it. For example, FlexiView is successful in reducing the planning time in comparison to zooming+scrolling but did not improve the navigation time. We observed that the design of the menus, buttons and the slider could be more efficient. Generally, we need to analyze our data even deeper to find ways to amplify the strengths of FlexiView and overcome its weaknesses. Information

navigation is one of the aspects of requirements modeling tools that affect usability. In the next steps, we can study other aspects, e.g., other screen types and input devices, recent human-computer interaction techniques such as augmented reality and virtual reality techniques, collaboration techniques, and machine-learning prediction methods.

Bibliography

[AKGC14] Pablo Oliveira Antonino, Thorsten Keuler, Nicolas Germann, and Brian Cronauer. A non-invasive approach to trace architecture design, requirements specification and agile artifacts. In *23rd Australian Software Engineering Conference (ASWEC)*, pages 220–229, 2014.

[BAL15] Edna Braun, Daniel Amyot, and Timothy C. Lethbridge. Generating software documentation in use case maps from filtered execution traces. In *17th International SDL Forum of Model-Driven Engineering for Smart Cities*, pages 177–192. Springer, 2015.

[Bev95] Nigel Bevan. Measuring usability as quality of use. *Software Quality Journal*, 4(2):115–130, 1995.

[BH94] Benjamin B. Bederson and James D. Hollan. Pad++: A zooming graphical interface for exploring alternate

interface physics. In *Proceedings of the 7th Annual ACM Symposium on User Interface Software and Technology (UIST)*, pages 17–26. ACM, 1994.

[BKZ10] Andrew Begel, Yit Phang Khoo, and Thomas Zimmermann. Codebook: discovering and exploiting relationships in software repositories. In *Proceeding of the ACM/IEEE 32nd International Conference on Software Engineering (ICSE 2010)*, pages 125–134. ACM, 2010.

[BMS+08] Chris Bennett, Del Myers, Margaret-Anne Storey, Daniel M German, David Ouellet, Martin Salois, and Philippe Charland. A survey and evaluation of tool features for understanding reverse-engineered sequence diagrams. *Journal of Software Maintenance and Evolution: Research and Practice*, 20(4):291–315, 2008.

[BSN12] Brian Berenbach, Florian Schneider, and Helmut Naughton. The use of a requirements modeling language for industrial applications. In *Proceedings of the 20th IEEE International Requirements Engineering Conference (RE '12)*, pages 285–290. IEEE Computer Society, 2012.

[CHH07] Jane Cleland-Huang and Rafal Habrat. Visual support in automated tracing. In *Proceedings of the Second International Workshop on Requirements Engineering Visualization (REV 2007)*. IEEE, 2007.

[CJLGG09] John R. Cooper Jr, Seok-Won Lee, Robin A. Gandhi, and Orlena Gotel. Requirements engineering visualization: a survey on the state-of-the-art. In *Proceedings of the fourth International Workshop on Requirements Engineering Visualization (REV)*, pages 46–55. IEEE, 2009.

[CKB09] Andy Cockburn, Amy Karlson, and Benjamin B. Bederson. A review of overview+detail, zooming, and Focus+Context interfaces. *ACM Computing Surveys (CSUR)*, 41(1):1–31, 2009.

[CKI88] Bill Curtis, Herb Krasner, and Neil Iscoe. A field study of the software design process for large systems. *Communications of the ACM*, 31(11):1268–1287, 1988.

[Cre13] John W. Creswell. *Research design: Qualitative, quantitative, and mixed methods approaches*. Sage publications, 2013.

[CSR+03] Mary Czerwinski, Greg Smith, Tim Regan, Brian Meyers, George Robertson, and Gary Starkweather. Toward characterizing the productivity benefits of very large displays. In *Proceedings of the IFIP TC13 International Conference on Human-Computer Interaction (INTERACT '03)*, pages 9–16. IOS Press, 2003.

[CVDMZ07] Bas Cornelissen, Arie Van Deursen, Leon Moonen, and Andy Zaidman. Visualizing testsuites to aid in software understanding. In *Proceedings of the 11th European Conference on Software Maintenance and Reengineering (CSMR'07)*, pages 213–222. IEEE, 2007.

[DAMR07] Brian De Alwis, Gail C. Murphy, and Martin P. Robillard. A comparative study of three program exploration tools. In *15th IEEE International Conference on Program Comprehension (ICPC '07)*, pages 103–112. IEEE, 2007.

[DC06] Daniela Damian and James Chisan. An empirical study of the complex relationships between requirements engineering processes and other processes that lead to payoffs in productivity, quality, and risk management. *IEEE Transactions on Software Engineering*, 32(7):433–453, 2006.

[DDH+06] Alan Davis, Oscar Dieste, Ann Hickey, Natalia Juristo, and Ana M. Moreno. Effectiveness of requirements elicitation techniques: Empirical results derived from a systematic review. In *Proceeding of the 14th IEEE International Requirements Engineering Conference (RE '06)*, pages 179–188. IEEE, 2006.

[Dek05] Uri Dekel. Supporting distributed software design meetings: what can we learn from co-located meetings? In *ACM SIGSOFT Software Engineering Notes*, number 4, pages 1–7. ACM, 2005.

[dGNA+11] Juan M Carrillo de Gea, Joaquín Nicolás, José L Fernández Alemán, Ambrosio Toval, Christof Ebert, and Aurora Vizcaíno. Requirements engineering tools. *IEEE Software*, 28(4):86–91, 2011.

[dGNA+12] Juan M. Carrillo de Gea, Joaquín Nicolás, José L Fernández Alemán, Ambrosio Toval, Christof Ebert, and Aurora Vizcaíno. Requirements engineering tools: Capabilities, survey and assessment. *Information and Software Technology*, 54(10):1142–1157, 2012.

[DH07] Uri Dekel and James D. Herbsleb. Notation and representation in collaborative object-oriented design: an observational study. In *ACM SIGPLAN Notices*, pages 261–280. ACM, 2007.

[dSAdO05] Sergio Cozzetti B. de Souza, Nicolas Anquetil, and Káthia M. de Oliveira. A study of the documentation essential to software maintenance. In *Proceedings of the 23rd Annual International Conference on Design of Communication: Documenting & Designing for Pervasive Information*, pages 68–75. ACM, 2005.

[dSR08] Cleidson R. B. de Souza and David F. Redmiles. An empirical study of software developers' management of dependencies and changes. In *Proceedings of the 30th International Conference on Software Engineering (ICSE '08)*, pages 241–250. ACM, 2008.

[Ead84] Peter Eades. A heuristic for graph drawing. *Congresses Numerantium*, 42(3):149–160, 1984.

[EGK+04] John Ellson, Emden R. Gansner, Eleftherios Koutsofios, Stephen C North, and Gordon Woodhull. Graphviz and Dynagraph – static and dynamic graph drawing tools. In *Graph Drawing Software*, pages 127–148. Springer, 2004.

[Eic08] Holger Eichelberger. Automatic layout of UML use case diagrams. In *Proceedings of the 4th International Symposium on Software Visualization*, pages 105–114. ACM, 2008.

[Eig03] Markus Eiglsperger. *Automatic layout of UML class diagrams: a topology-shape-metrics approach.* PhD thesis, Universität Tübingen, 2003.

[ELC+98] Steve Easterbrook, Robyn Lutz, Richard Covington, John Kelly, Yoko Ampo, and David Hamilton. Experiences using lightweight formal methods for requirements modeling. *IEEE Transaction of Software Engineering*, 24(1):4–14, 1998.

[ESSD08] Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. Selecting empirical methods for software engineering research. In *Guide to Advanced Empirical Software Engineering*, pages 285–311. Springer, 2008.

[FB95] George W. Furnas and Benjamin B. Bederson. Space-scale diagrams: Understanding multiscale interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '95)*, pages 234–241. ACM Press/Addison-Wesley Publishing Co., 1995.

[FD10] Mathias Frisch and Raimund Dachselt. Off-screen visualization techniques for class diagrams. In *Proceedings of the 5th International Symposium on Software Visualization*, pages 163–172. ACM, 2010.

[FD13] Mathias Frisch and Raimund Dachselt. Visualizing offscreen elements of node-link diagrams. *Information Visualization*, 12(2):133–162, 2013.

[FDB08] Mathias Frisch, Raimund Dachselt, and Tobias Brückmann. Towards seamless semantic zooming techniques for UML diagrams. In *Proceedings of the ACM Symposium on Software Visualization,*, pages 207–208, 2008.

[FHH00] Erik Frøkjær, Morten Hertzum, and Kasper Hornbæk. Measuring usability: Are effectiveness, efficiency, and satisfaction really correlated? In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '00)*, pages 345–352. ACM, 2000.

[FL02] Andrew Forward and Timothy C. Lethbridge. The relevance of software documentation, tools and technologies: A survey. In *Proceedings of the 2002 ACM*

Symposium on Document Engineering (DocEng '02), pages 26–33. ACM, 2002.

[FR91] Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11):1129–1164, 1991.

[Fur86] George W. Furnas. Generalized fisheye views. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '86)*, pages 16–23. ACM, 1986.

[FvH10] Hauke Fuhrmann and Reinhard von Hanxleden. Taming graphical modeling. In *Proceedings of the 13th International Conference on Model Driven Engineering Languages and Systems (MODELS'10)*, pages 196–210. Springer-Verlag, 2010.

[GAG17] Parisa Ghazi, Zahra Shakeri Hossein Abad, and Martin Glinz. Choosing requirements for experimentation with user interfaces of requirements modeling tools. In *Proceeding of the 25th IEEE International Requirements Engineering Conference (RE '17)*, pages 462–463. IEEE Computer Society, 2017.

[GBJ02] Martin Glinz, Stefan Berner, and Stefan Joos. Object-oriented modeling with ADORA. *Information System*, 27(6):425–444, 2002.

[GG16] Parisa Ghazi and Martin Glinz. An exploratory study on user interaction challenges when handling interconnected requirements artifacts of various sizes. In *Proceeding of the 24th IEEE International Requirements Engineering Conference (RE '16)*, pages 76–85. IEEE, 2016.

[GG17a] Parisa Ghazi and Martin Glinz. Challenges of working with artifacts in requirements engineering and software engineering. *Requirements Engineering Journal (REJ)*, 22(3):359–385, 2017.

[GG17b] Parisa Ghazi and Martin Glinz. Imitgraphs: Towards faster usability tests of graphical model manipulation techniques. In *Proceeding of the 9th International Workshop on Modeling in Software Engineering (MiSE@ICSE2017)*, pages 61–67. IEEE, 2017.

[GG18] Parisa Ghazi and Martin Glinz. An experimental comparison of two navigation techniques for requirements modeling tools. In *Proceeding of the 26th IEEE International Requirements Engineering Conference (RE '18)*. IEEE, 2018.

[GGM+13] Golara Garousi, Vahid Garousi, Mahmoud Moussavi, Guenther Ruhe, and Brian Smith. Evaluating usage and quality of technical software documentation: An empirical study. In *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering (EASE '13)*, pages 24–35. ACM, 2013.

[Gli10] Martin Glinz. Very lightweight requirements modeling. In *Proceedings of the 18th IEEE International Requirements Engineering Conference (RE '10)*, pages 385–386. IEEE, 2010.

[GSG15] Parisa Ghazi, Norbert Seyff, and Martin Glinz. FlexiView: A magnet-based approach for visualizing requirements artifacts. In *Proceedings of the 21st International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ '15)*, pages 262–269. Springer, 2015.

[HKWB04] Matthias Hoffmann, Nikolaus Kuhn, Matthias Weber, and Margot Bittner. Requirements for requirements management tools. In *Proceedings of the 12th IEEE International Requirements Engineering Conference (RE '04)*, pages 301–308. IEEE Computer Society, 2004.

[Hol05] Andreas Holzinger. Usability engineering methods for software developers. *Communications of the ACM*, 48(1):71–74, 2005.

[IEE90] IEEE. *IEEE standard glossary of software engineering terminology.* IEEE Std. 610.12, 1990.

[Int98] International Organization for Standardization. *ISO 9241-11: Ergonomic requirements for office work with visual display terminals (VDTs): Part 11guidance on usability.* 1998.

[Int18] International Organization for Standardization. *ISO 9241-11:2018: Ergonomics of human-system interaction – Part 11: Usability: Definitions and concepts.* 2018.

[KDoD+02] Lena Karlsson, Åsa Dahlstedt, Johan Natt och Dag, Björn Regnell, and Anne Persson. Challenges in market-driven requirements engineering - An industrial interview study. In *Proceedings of the 8th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ '02)*, pages 37–49, 2002.

[KDvV02] Henk Koning, Claire Dormann, and Hans van Vliet. Practical guidelines for the readability of IT-architecture diagrams. In *Proceedings of the 20th Annual International Conference on Computer Documentation (SIGDOC '02)*, pages 90–99. ACM, 2002.

[KH10] Nigel King and Christine Horrocks. *Interviews in qualitative research.* Sage, 2010.

[KM07] Huzefa Kagdi and Jonathan I. Maletic. Onion graphs for Focus+Context views of UML class diagrams. In *Proceedings of the 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT 2007)*, pages 80–87. IEEE, 2007.

[KP02] Barbara Kitchenham and Shari Lawrence Pfleeger. Principles of survey research: part 5: populations

and samples. *ACM SIGSOFT Software Engineering Notes*, 27(5):17–20, 2002.

[LA94] Ying K. Leung and Mark D. Apperley. A review and taxonomy of distortion-oriented presentation techniques. *ACM Transactions on Computer Human Interaction*, 1(2):126–160, 1994.

[Lay18] David Lay. *Designing and Conducting Controlled Experiments on an Experimental Requirements Modeling Tool for Evaluating ImitGraphs*. Bachelor Thesis, Department of Informatics, University of Zurich, 2018.

[Lew06] James R. Lewis. Usability testing. *Handbook of Human Factors and Ergonomics*, 12:e30, 2006.

[Lik32] Rensis Likert. A technique for the measurement of attitudes. *Archives of Psychology*, 1932.

[Lin99] Kurt R. Linberg. Software developer perceptions about software project failure: A case study. *Journal of Systems and Software*, 49(2-3):177–192, 1999.

[Lis15] Olga Liskin. How artifacts support and impede requirements communication. In *Proceedings of the 21st International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ '15)*, pages 132–147. Springer, 2015.

[LM10] Heidi Lam and Tamara Munzner. A guide to visual multi-level interface design from synthesis of empirical study evidence. *Synthesis Lectures on Visualization*, 1(1):1–117, 2010.

[LMW+15] Lars Lischke, Sven Mayer, Katrin Wolf, Niels Henze, Albrecht Schmidt, Svenja Leifert, and Harald Reiterer. Using space: Effect of display size on users' search performance. In *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems (CHI '15)*, pages 1845–1850. ACM, 2015.

[LSF03] Timothy C. Lethbridge, Janice Singer, and Andrew Forward. How software engineers use documentation: The state of the practice. *IEEE Software*, 20(6):35–39, 2003.

[LSS05] Timothy C. Lethbridge, Susan Elliott Sim, and Janice Singer. Studying software engineers: Data collection techniques for software field studies. *Empirical Software Engineering*, 10(3):311–341, 2005.

[MAB+14] Gunter Mussbacher, Daniel Amyot, Ruth Breu, Jean-Michel Bruel, Betty HC Cheng, Philippe Collet, Benoit Combemale, Robert B France, Rogardt Heldal, James Hill, et al. The relevance of model-driven engineering thirty years from now. In *International Conference on Model Driven Engineering Languages and Systems*, pages 183–200. Springer, 2014.

[May99] Deborah J. Mayhew. The usability engineering lifecycle. In *Extended Abstracts on Human Factors in Computing Systems (CHI EA '99)*, pages 147–148. ACM, 1999.

[MBD+10] Nicolas Mangano, Alex Baker, Mitch Dempsey, Emily Navarro, and André van der Hoek. Software design sketching with Calico. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE '10)*, pages 23–32. ACM, 2010.

[MF13] Sebastian C. Müller and Thomas Fritz. Stakeholders' information needs for artifacts and their dependencies in a real world context. In *Proceeding of the 29th IEEE International Conference on Software Maintenance (ICSM)*, pages 290–299. IEEE, 2013.

[MHP00] Brad Myers, Scott E. Hudson, and Randy Pausch. Past, present, and future of user interface software tools. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 7(1):3–28, 2000.

[MM07] Wendy L. Martinez and Angel R. Martinez. *Computational statistics handbook with MATLAB*. CRC press, 2007.

[MSG+07] Neil A. M. Maiden, Norbert Seyff, Paul Grünbacher, Omo Otojare, and Karl Mitteregger. Determining

stakeholder needs in the workplace: How mobile technologies can help. *IEEE Software*, 24(2):46–52, 2007.

[Nie94] Jakob Nielsen. *Usability engineering*. Elsevier, 1994.

[Oat05] Briony J. Oates. *Researching Information Systems and Computing*. Sage, 2005.

[Oeh13] Andreas Oehlke. *Learning Libgdx Game Development*. Packt Publishing, 2013.

[OvdHS+10] Harold Ossher, André van der Hoek, Margaret-Anne Storey, John Grundy, and Rachel Bellamy. Flexible modeling tools (Flexitools 2010). In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, pages 441–442. ACM, 2010.

[OvdHS+11] Harold Ossher, André van der Hoek, Margaret-Anne Storey, John Grundy, Rachel Bellamy, and Marian Petre. Flexible modeling tools (Flexitools 2011). In *Proceedings of the 33rd International Conference on Software Engineering*, pages 1192–1193. ACM, 2011.

[Par13a] Ian Parberry. *Introduction to Game Physics with Box2D*. CRC Press, Inc., 1st edition, 2013.

[Par13b] Paul Parsons. *Cognitive Activity Support Tools: Design of the Visual Interface*. PhD thesis, The University of Western Ontario, 2013.

[PB13] Dennis Pagano and Bernd Brügge. User involvement in software evolution practice: a case study. In *Proceedings of the 35th International Conference on Software Engineering (ICSE '13)*, pages 953–962. IEEE, 2013.

[Pet09] Marian Petre. Insights from expert software design practice. In *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering ESEC/FSE '09*, pages 233–242. ACM, 2009.

[Poh10] Klaus Pohl. *Requirements engineering: fundamentals, principles, and techniques*. Springer Publishing Company, Incorporated, 2010.

[PPCP12] Cyprien Pindat, Emmanuel Pietriga, Olivier Chapuis, and Claude Puech. Jellylens: Content-aware adaptive lenses. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology (UIST '12)*, pages 261–270. ACM, 2012.

[PRVPB12] Javier Portillo-Rodríguez, Aurora Vizcaíno, Mario Piattini, and Sarah Beecham. Tools used in global software engineering: A systematic mapping review. *Information and Software Technology*, 54(7):663–685, 2012.

[RCB+05] George Robertson, Mary Czerwinski, Patrick Baudisch, Brian Meyers, Daniel Robbins, Greg Smith, and Desney Tan. The large-display user experience. 25(4):44–51, 2005.

[RCN12] Sandeep Reddivari, Zhangji Chen, and Nan Niu. ReCVisu: A tool for clustering-based visual exploration of requirements. In *Proceedings of the 20th IEEE International Requirements Engineering Conference (RE '12)*, pages 327–328. IEEE, 2012.

[RK14] Jeffrey M. Rzeszotarski and Aniket Kittur. Kinetica: Naturalistic multi-touch data visualization. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*, pages 897–906. ACM, 2014.

[RMG07] Tobias Reinhard, Silvio Meier, and Martin Glinz. An improved fisheye zoom algorithm for visualizing and editing hierarchical models. In *Proceedings of the second International Workshop on Requirements Engineering Visualization (REV 2007)*. IEEE, 2007.

[RMS+08] Tobias Reinhard, Silvio Meier, Reinhard Stoiber, Christina Cramer, and Martin Glinz. Tool support for the navigation in graphical models. In *Proceedings of the 30th International Conference on Software Engineering (ICSE '08)*, pages 823–826. ACM, 2008.

[Rob02] Colin Robson. *Real World Research*. Blackwell Publishing. Malden, second edition, 2002.

[RTKM12] Tobias Roehm, Rebecca Tiarks, Rainer Koschke, and Walid Maalej. How do professional developers comprehend software? In *Proceedings of the 34th International Conference on Software Engineering (ICSE '12)*, pages 255–265. IEEE Press, 2012.

[SB94] Manojit Sarkar and Marc H. Brown. Graphical fisheye views. *Communications of the ACM*, 37(12):73–83, 1994.

[Sch11] Christoph Daniel Schulze. *Optimizing automatic layout for data flow diagrams.* PhD thesis, Diploma thesis, Christian-Albrechts-Universität zu Kiel, Department of Computer Science, 2011.

[Sea08] Carolyn B. Seaman. Qualitative methods. In *Guide to advanced empirical software engineering*, pages 35–62. Springer-Verlag, 2008.

[SF08] Andre Suslik Spritzer and Carla M. D. S. Freitas. A physics-based approach for interactive manipulation of graph visualizations. In *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI '08)*, pages 271–278. ACM, 2008.

[Shn10] Ben Shneiderman. *Designing the user interface: strategies for effective human-computer interaction.* Pearson Education India, 2010.

[Som01] Ian Sommerville. Software documentation. *Software Engineering*, 2:143–154, 2001.

[Spö15] Miro Spönemann. *Graph Layout Support for Model-Driven Engineering*. BoD–Books on Demand, 2015.

[TA13] Thomas Tullis and William Albert. *Measuring the User Experience: Collecting, Analyzing, and Presenting Usability Metrics*. Morgan Kaufmann Publishers Inc., USA, 2013.

[TNLJ+14] Miguel A. Teruel, Elena Navarro, Víctor López-Jaquero, Francisco Montero, and Pascual González. A CSCW requirements engineering CASE tool: development and usability evaluation. *Information and Software Technology*, 56(8):922–949, 2014.

[TR04] Tuure Tuunanen and Matti Rossi. Engineering a method for wide audience requirements elicitation and integrating it to software development. In *Proceedings of the 37th Hawaii International Conference on System Sciences HICSS*. IEEE Computer Society, 2004.

[Win07] Stefan Winkler. Information flow between requirement artifacts. Results of an empirical study. In *Proceedings of the 13th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ '07)*, pages 232–246. Springer, 2007.

[WMMR05] Roel Wieringa, Neil Maiden, Nancy Mead, and Colette Rolland. Requirements engineering paper classification and evaluation criteria: A proposal and a

discussion. *Requirements Engineering Journal (REJ)*, 11(1):102–107, 2005.

[WP10] Stefan Winkler and Jens Pilgrim. A survey of traceability in requirements engineering and model-driven development. *Software System Model*, 9(4):529–565, 2010.

[WRH+00] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, 2000.

[WSG12a] Dustin Wüest, Norbert Seyff, and Martin Glinz. Flexible, lightweight requirements modeling with FlexiSketch. In *Proceedings of the 20th IEEE International Requirements Engineering Conference (RE '12)*, pages 323–324. IEEE, 2012.

[WSG12b] Dustin Wüest, Norbert Seyff, and Martin Glinz. FlexiSketch: A mobile sketching tool for software modeling. In *Proceedings of the Fourth International Conference on Mobile Computing, Applications and Services (MobiCASE)*, pages 225–244. Springer-Verlag, 2012.

[WSG13] Dustin Wüest, Norbert Seyff, and Martin Glinz. Semi-automatic generation of metamodels from model sketches. In *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering (ASE'13)*, pages 664–669, 2013.

[WSG15] Dustin Wüest, Norbert Seyff, and Martin Glinz. Sketching and notation creation with FlexiSketch Team: Evaluating a new means for collaborative requirements elicitation. In *Proceedings of the 23rd IEEE International Requirements Engineering Conference (RE '15)*, pages 186–195. IEEE, 2015.

[ZGYS+15] Junji Zhi, Vahid Garousi-Yusifoğlu, Bo Sun, Golara Garousi, Shawn Shahnewaz, and Guenther Ruhe. Cost, benefits and quality of software development documentation: A systematic mapping. *Journal of Systems and Software*, 99:175–198, 2015.

Appendix A
Publications

This appendix presents the list of publications on which this cumulative dissertation is built. Publications that are not included in the dissertation are marked as such.

A.1 Conference Papers

[GG18a] Parisa Ghazi and Martin Glinz. An Experimental Comparison of Two Navigation Techniques for Requirements Modeling Tools. In *Proceedings of the 26th IEEE International Requirements Engineering Conference (RE'18)*, 2018.

[GG18b] Parisa Ghazi and Martin Glinz. FlexiView Experimental Tool: Fair and Detailed Usability Tests for Requirements Modeling Tools. In *Proceedings of the 26th IEEE International Requirements Engineering Conference (RE'18)*, 2018.

[GG17a] Parisa Ghazi and Martin Glinz. ImitGraphs: Towards Faster Usability Tests of Graphical Model Manipulation Techniques. In *Proceedings of the 9th IEEE/ACM International Workshop on Modelling in Software Engineering ((MiSE@ICSE)*, pages 61–67, 2017.

[GSG17b] Parisa Ghazi, Zahra Shakeri Hossein Abad, and Martin Glinz. Choosing Requirements for Experimentation with User Interfaces of Requirements Modeling Tools. In *Proceedings of the 25th IEEE International Requirements Engineering Conference (RE'17)*, pages 462–463, 2017.

[SKGGRS17c] Zahra Shakeri Hossein Abad, Oliver Karras, Parisa Ghazi, Martin Glinz, Guenther Ruhe, and Kurt Schneider. What Works Better? A Study of Classifying Requirements. In *Proceedings of the 25th IEEE International Requirements Engineering Conference (RE'17)*, pages 496–501, 2017.(**Not included in this thesis.**)

[GG16] Parisa Ghazi and Martin Glinz. An Exploratory Study on User Interaction Challenges When Handling Interconnected Requirements Artifacts of Various Sizes. In *Proceedings of the 24th IEEE International Requirements Engineering Conference (RE'16)*, pages 76–85, 2016.

[GSG15a] Parisa Ghazi, Norbert Seyff, and Martin Glinz. FlexiView: A Magnet-Based Approach for Visualizing Requirements Artifacts. In *Proceedings of the 21st International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'15)*, pages 262–269, 2015.

A.2 Journal articles

[GG15c] Parisa Ghazi and Martin Glinz. Challenges of working with artifacts in requirements engineering and software engineering. Requirements Engineering, 22, 3, pages 359–385, 2017.

A.3 PhD Symposium

[Ghazi15] Parisa Ghazi. A Magnet-and-Spring Based Visualization Technique for Enhancing the Manipulation of Requirements. In *Doctoral Symposium of the 23rd IEEE International Requirements Engineering Conference (RE'15)*, 2015. (**Not included in this thesis.**)

Curriculum Vitae

Name:	Parisa Ghazi
Date of Birth:	September 20, 1986
Citizenship:	Iran

2014 - 2018	Assistant and doctoral student at the *University of Zurich*, Switzerland
2011 - 2013	Master of Science in Computer Science (Software Engineering) at the *Technical University of Madrid*, Spain
2004 - 2009	Bachelor in Computer Engineering (Software Engineering) at the *Islamic Azad University (Tehran North Branch)*, Iran
2001 - 2004	High school, Moalem, Tehran, Iran